UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Implementación de una Interfaz de Datos RS-232 Inalámbrica para el Radar Láser Hokuyo URG-04LX-UG01

Trabajo de graduación presentado por Mario Andrés Búrbano Castro para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala
2012

Implementación de una Interfaz de Datos RS-232 Inalámbrica para el Radar Láser Hokuyo URG-04LX-UG01

UNIVERSIDAD DEL VALLE DE GUATEMALA

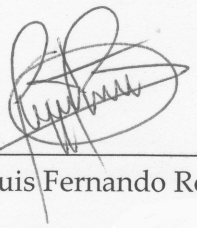
Facultad de Ingeniería

Implementación de una Interfaz de Datos RS-232 Inalámbrica para el Radar Láser Hokuyo URG-04LX-UG01

Trabajo de graduación presentado por Mario Andrés Búrbano Castro para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala
2012

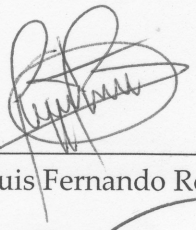
Vo. Bo.:



(f) _____

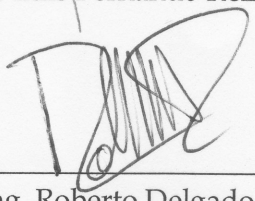
Ing. Luis Fernando Reina

Tribunal:



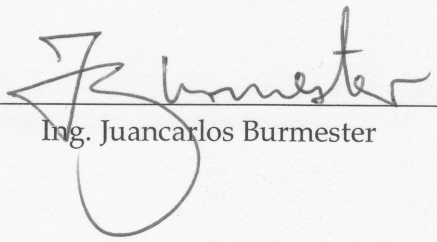
(f) _____

Ing. Luis Fernando Reina



(f) _____

Ing. Roberto Delgado



(f) _____

Ing. Juancarlos Burmester

Guatemala, 21 de junio del 2012.

PREFACIO

Este trabajo de graduación surge originalmente con el propósito de desarrollar un sistema de reconocimiento de entorno para el cuadricóptero del megaproyecto SEEQ: *Smart Environment Exploring Quadcopter* de la Universidad del Valle de Guatemala (Búrbano, Saravia, Sandoval, Suazo, & Morales, 2011). Sin embargo, el objetivo principal de esta tesis es contribuir al desarrollo de futuros megaproyectos relacionados a la robótica y exploración a través de la implementación de un sistema de reconocimiento de entorno de largo alcance y alta resolución.

Además, este trabajo de graduación busca continuar con la línea de investigación desarrollada en los departamentos de Ingeniería Electrónica y Mecatrónica de la Universidad del Valle de Guatemala relacionada a la robótica asistencial y la exploración de ambientes desconocidos.

Quiero agradecer a Dios, por darme la oportunidad de cumplir mi meta de ser ingeniero mecatrónico; a mi familia, por su apoyo moral y económico incondicional durante el desarrollo de mi carrera profesional; a mi asesor, Ing. Luis Fernando Reina, por su apoyo y consejo; a los administradores de los laboratorios del departamento de Ingeniería Electrónica y Mecatrónica, Ing. Estuardo Mancio y Willy Reyes, por su constante apoyo durante la etapa de implementación de esta tesis y a mis compañeros, José Morales, Daniel Suazo, Melisa Sandoval y Roberto Saravia, por su valioso aporte a este trabajo de graduación.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

Este trabajo de graduación presenta la implementación de una interfaz de datos inalámbrica RS-232 para el radar láser HokuyoURG-04LX-UG01. La interfaz permite utilizar el radar en aplicaciones robóticas donde se requiera la implementación de un sistema de reconocimiento de entorno de largo alcance y alta resolución.

El radar láser utilizado posee un área de detección de 240°, tiene una resolución de 1 mm, una resolución angular de 0.35°y un rango de medición comprendido entre 20 cm y 5.60 m. Además, el radar es capaz de transmitir las distancias medidas hasta 10 veces por segundo (cada 100 ms).

Para controlar el radar láser fue necesario implementar un anfitrión USB dedicado (*USB Embedded Host* en inglés) en un microcontrolador compatible con el estándar USB On-The-Go (OTG); se utilizó el microcontrolador PIC24FJ64GB002. Se utilizaron módulos de radio frecuencia *XBee Pro* de 60 mW para la transmisión inalámbrica de los datos utilizando el protocolo de comunicación RS-232.

Por otro lado, se desarrolló una interfaz gráfica en un lenguaje de alto nivel (C#) que permite visualizar gráficamente los datos del radar láser y configurar sus parámetros de funcionamiento. Dichos parámetros son almacenados en la memoria no volátil del microcontrolador para adaptar la interfaz de datos desarrollada a la mayoría de aplicaciones robóticas.

Por último, los resultados obtenidos en este trabajo de graduación fueron publicados como código abierto en el sitio de internet “Simple-URG Viewer” (Búrbano, Wireless RS-232 Data Interface for Simple-URG, 2012) desarrollado como documentación de referencia adicional para la aplicación.

I. INTRODUCCIÓN

Un robot es, por definición, una máquina capaz de interactuar con su entorno para realizar alguna tarea o resolver algún problema. Esto implica que debe ser capaz de adaptar sus movimientos en base a las características físicas de su entorno y los objetos que haya en él. Por esta razón, los robots requieren tener conocimiento del entorno que los rodea.

El reconocimiento de entorno es un aspecto muy importante en la mayoría de aplicaciones robóticas ya que provee información acerca del entorno del robot. Esta información puede ser utilizada para la creación de mapas globales del entorno, implementación de algoritmos de navegación y detección de obstáculos, entre otras. Para esto se utilizan diversos tipos de sensores que varían en cuanto a su complejidad y uso. El reconocimiento de entorno puede ser implementado a través de métodos como la estereovisión, omnivisión o sensores como cámaras 3D, cámaras de profundidad (*depth cameras* en inglés), cámaras de tiempo de vuelo (*ToF cameras* en inglés) y telémetros (también llamados radares o *rangefinders*) láser y ultrasónicos.

Por lo anterior, se decidió utilizar el radar láser Hokuyo URG-04LX-UG01 para la implementación de un sistema de reconocimiento de entorno ya que, por sus características de resolución, exactitud, alcance y frecuencia de muestreo, resulta idóneo para una aplicación de ese tipo. Sin embargo, el radar láser adquirido cuenta únicamente con una interfaz de datos USB 2.0. Según el estándar USB, los dispositivos periféricos, en este caso el radar láser, pueden comunicarse únicamente con un *anfitrión USB* (*Host USB* por su nombre en inglés) que controla el bus de comunicación; el *Host USB* más común son las computadoras. Esto representa un inconveniente ya que en la mayoría de aplicaciones no resulta factible incorporar una computadora al diseño de un vehículo robótico debido a limitantes de peso, espacio y costo.

Por esta razón, surge la necesidad de implementar una interfaz de datos para el radar láser compatible con alguno de los protocolos de comunicación implementados comúnmente en los microcontroladores (e.g., RS-232, I^2C, SPI) que sea capaz de transmitir los datos del radar inalámbricamente. Se eligió el estándar RS-232 para la implementación de la interfaz de datos debido a su simplicidad de uso y fácil integración con una computadora u otros microcontroladores. Sin embargo, surge la limitante de la comunicación punto-punto con un solo dispositivo simultáneamente.

En el capítulo 1 se explica a detalle el proceso de implementación del *Host USB* dedicado en el microcontrolador. Además, se incluye una descripción acerca de la estructura de programación utilizada y se explican cada uno de los estados de la máquina de estados finitos implementada.

Luego, en el capítulo 2 se exponen los algoritmos utilizados para la decodificación de los datos del radar. También se explican la detección y el manejo de errores en la transmisión de los datos.

Posteriormente, en el capítulo 3 se explica el sistema de comunicación inalámbrica implementado y se explica el algoritmo de la aplicación desarrollada en el lenguaje de alto nivel C# que permite visualizar la información del radar láser en tiempo real y modificar sus parámetros de funcionamiento.

Finalmente, en el capítulo 4 se presenta el proceso de integración de la interfaz de datos al megaproyecto SEEQ: *Smart Environment Exploring Quadcopter* (Búrbano, Saravia, Sandoval, Suazo, & Morales, 2011).

II. OBJETIVOS

A. Objetivo general

- Implementar una interfaz de datos inalámbrica RS-232 para el radar láser Hokuyo URG-04LX-UG01 que permita utilizarlo en un sistema de reconocimiento de entorno.

B. Objetivos específicos

- Implementar en un microcontrolador un anfitrión USB dedicado para dispositivos USB del tipo CDC (*Communications Device Class*) que permita controlar el sensor láser Hokuyo URG-04LX-UG01.

- Implementar un sistema de comunicación inalámbrica que permita controlar el radar láser remotamente utilizando el protocolo RS-232.

- Desarrollar una interfaz gráfica en un lenguaje de alto nivel que permita visualizar los datos del radar láser en tiempo real y configurar sus parámetros de funcionamiento.

III. MARCO TEÓRICO

A. Sensores medidores de distancia

Los sensores medidores de distancia poseen un transductor que genera una señal analógica eléctrica relacionada con la distancia existente entre el sensor y el objeto de medición más cercano. Esto se realiza usualmente mediante sensores fotoeléctricos, inductivos o ultrasónicos (Carletti, 2010).

1. Sensores fotoeléctricos Los sensores fotoeléctricos son dispositivos electrónicos que responden a un cambio en la intensidad de la luz. Contienen un emisor que emite un haz de luz que viaja hasta rebotar con un objeto que se interpone; luego el haz de luz cambia de dirección y viaja de regreso hacia el receptor del sensor. Esto permite censar el objeto y calcular la distancia hacia el emisor. La mayoría de formas de censado se basan en este mismo principio de funcionamiento (Carletti, 2010).

Este tipo de sensores son utilizados comúnmente para la detección de objetos, formas, colores y medición de distancias. Los sensores fotoeléctricos se diferencian por la fuente de luz que utilizan; las más comunes son LED, infrarroja y láser (Carletti, 2010).

a. Telémetros láser de barrido (*Scanning laser range finders*)

1) Descripción Los telémetros o radares láser funcionan como un sensor de alta resolución para aplicaciones robóticas. Son de gran utilidad en la mayoría de aplicaciones debido a su alta resolución, exactitud y frecuencia de muestreo.

En muchas aplicaciones se requiere el poder computacional de un microprocesador en el procesamiento de los datos para aprovechar todas las características de este tipo de sensores debido a la gran cantidad de información que generan (Acroname Robotics, 2011).

Los radares láser proveen información acerca del entorno con una resolución de hasta 1 mm. Además, el consumo de potencia de este tipo de sensores es similar al de sensores muchos menos sofisticados como los capacitivos, ultrasónicos o infrarrojos (Acroname Robotics, 2011). Algunos de los fabricantes más comunes de este tipo de sensores son Hokuyo y SICK. La Figura 1 muestra dos modelos utilizados comúnmente en aplicaciones robóticas.

Figura 1: Telémetros láser Hokuyo URG-04LX y SICK LMS100

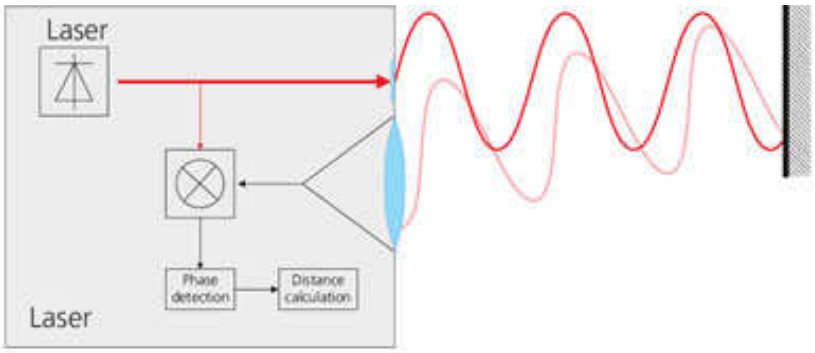
Fuente: (Acroname Robotics, 2011), (SICK, 2012)

2) Teoría de operación Los radares láser pueden ser pensados como sonares que utilizan una luz láser en vez del sonido para crear representaciones en 2 dimensiones de su entorno. Sin embargo, debido a que este tipo de sensores utilizan la luz en vez del sonido, son capaces de tomar mediciones de distancia mucho más rápido y con mayor exactitud.

El término "*de barrido*" describe la forma de medición del sensor: este toma una medición de distancia, luego se desplaza una fracción angular y toma una nueva medición; este proceso es repetido hasta completar una revolución. Sin embargo, debido a limitantes físicas y mecánicas, es común que estos sensores tengan una pequeña porción angular "ciega" en la cual no puedan realizar mediciones de distancia (Acroname Robotics, 2011).

Las mediciones son efectuadas utilizando el principio de medición de distancia basado en el desplazamiento de la fase del rayo láser o "*phase shift*". Según este principio, se utiliza un reloj interno incorporado en el sensor para medir el tiempo que tarda la luz láser en viajar hacia el objeto y regresar al sensor (véase Figura 2). Luego, se compara la onda emitida con la recibida para determinar el desplazamiento de la fase de ambas ondas y así poder calcular la distancia hacia el objeto más cercano.

Figura 2: Desplazamiento de fase en un sensor láser



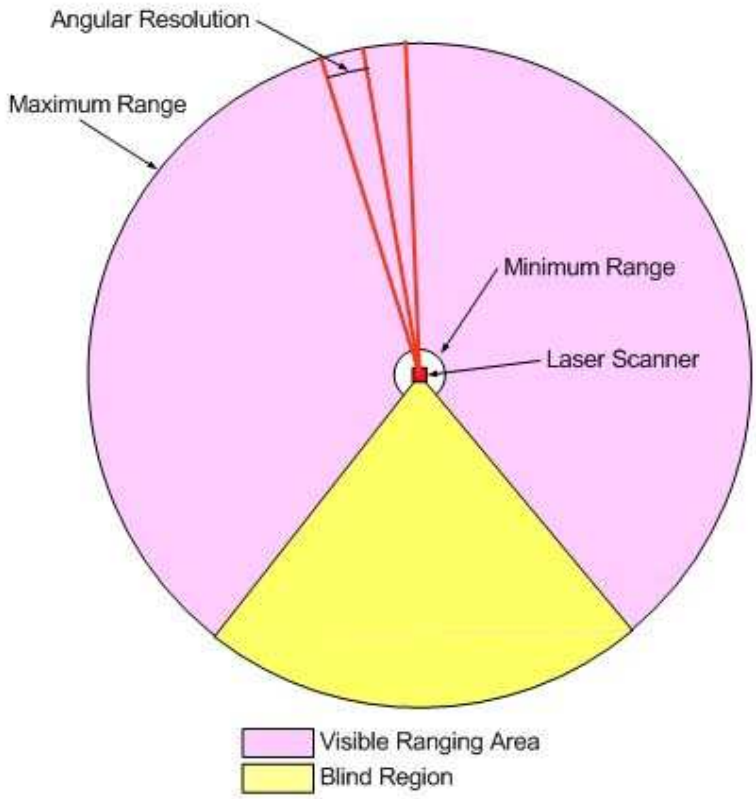
Fuente: (Metrology Resource Co., 2010)

El principio de medición descrito arriba permite obtener mediciones estables que no sean afectadas significativamente por el color, el material o la reflexión del objeto (Hokuyo, 2009).

3) Parámetros de funcionamiento Los datos que proveen este tipo de sensores usualmente se representan en coordenadas polares. El sensor representa el origen del plano polar y cada medición de distancia corresponde a un punto dentro del plano con coordenadas (r, θ).

Además, el "punto ciego" del sensor se encuentra en alguna fracción del mismo plano como lo muestra la Figura 3 (Acroname Robotics, 2011).

Figura 3: Funcionamiento de un radar láser



Fuente: (Acroname Robotics, 2011)

Los parámetros de funcionamiento de los radares láser se describen en la Tabla 1.

Tabla 1: Parámetros de funcionamiento de un radar láser

Parámetro	Descripción
Rango de detección (°)	Indica el tamaño, en grados, del sector circular en el cual el sensor puede tomar mediciones de distancia (véase Figura 3). Los radares láser usualmente poseen un rango de detección comprendido entre 240° - 270°.
Resolución (mm)	Indica el cambio más pequeño de distancia que puede medir el radar láser en una dirección en específico y está relacionada a la exactitud de las mediciones de distancia efectuadas. La resolución usualmente es de 1 – 3 mm.

Continuación Tabla 1

Parámetro	Descripción
Resolución angular (°)	Indica el desplazamiento angular más pequeño, en grados, que el radar láser puede efectuar entre mediciones consecutivas. Usualmente la resolución angular está comprendida entre 0.25° y 1° (véase Figura 3).
Frecuencia (Hz)	Indica el número de veces por segundo que el radar láser realiza un barrido completo de medición. Usualmente la frecuencia de los radares láser es igual o mayor a 10 Hz.
Rango mínimo (mm)	Representa la distancia más pequeña que el radar láser puede medir en una dirección en específico (véase Figura 3).
Rango máximo (m)	Representa la distancia más grande que el radar láser puede medir en una dirección en específico (véase Figura 3).

Fuente: (Acroname Robotics, 2011)

Además, la resolución angular determina el número de muestras tomadas en cada revolución del láser y la cantidad de datos que transmite el sensor según la Ecuación 1.

Ecuación 1: Número de muestras

$$No.\,de\ muestras = \frac{\acute{a}ngulo\ de\ barrido\ (°)}{resolución\ angular\ (°)}$$

4) Características físicas Los radares láser funcionan con alimentación de corriente directa que puede variar desde 5 hasta 30 VDC dependiendo del alcance de medición y las características del sensor. La mayoría de radares cuentan con una interfaz de datos serial (RS-232) o incluso USB en algunos modelos recientes.

Por otro lado, estas interfaces seriales de comunicación permiten, además de recibir los datos del sensor, modificar algunos parámetros de su funcionamiento como la resolución angular, la frecuencia y el ángulo de barrido, entre otros. Dentro de la estructura plástica del radar hay una cabeza giratoria sobre la cual está montada el sensor (Acroname Robotics, 2011).

El láser utilizado es regularmente uno de baja potencia que emite un haz de luz láser muy angosto que viaja hasta rebotar con algún objeto y luego es detectado por el receptor del radar. Este es el principio de funcionamiento de un sensor fotoeléctrico (véase Sensores fotoeléctricos).

B. Rayos láser

1. Diodos láser Los diodos láser son dispositivos semiconductores que emiten rayos láser. Este tipo de diodos son más eficientes y fiables que la mayoría de diodos LED. Además, poseen un tiempo de vida útil muy largo, su consumo de potencia es reducido y sus dimensiones físicas son pequeñas. Este tipo de diodos se diferencian por su longitud de onda (λ) medida en nanómetros (nm) y son utilizados en comunicación de datos por fibra óptica, lectores de CD/DVD, impresoras, digitalizadores, sensores, entre otros (Lasermet, Ltd, 2012).

Entre las longitudes de onda (λ) más comunes podemos mencionar: 405 nm, utilizada en los lectores de *Blu-ray* y *HD-DVD*; 670 nm, utilizada en la fabricación de punteros láser, 2004 nm, utilizada en la medición de dióxido de carbono (CO_2) y 785 nm, utilizada en medidores de distancia y lectores CD (Lasermet, Ltd, 2012).

2. Clasificación de los rayos láser Los rayos láser emitidos por los diodos láser están clasificados según su potencial para causar daños biológicos tomando en cuenta parámetros como: potencia disipada, longitud de onda de la radiación (λ), tiempo de exposición y área transversal del rayo láser (Universidad de Princeton, 2007).

Esta clasificación forma parte del estándar IEC60825-1 establecido por la Comisión Electrotécnica Internacional (IEC, por sus siglas en inglés). Dicha clasificación abarca desde la Clase 1, que incluye a los rayos láser que no producen ninguna radiación dañina al organismo, hasta la Clase 4, que agrupa a los que pueden producir daños

permanentes a la vista, quemaduras en la piel e incluso provocar incendios (Lasermet, Ltd, 2012). La clasificación descrita se muestra a continuación:

a. Clase 1 Son dispositivos de baja potencia considerados seguros bajo condiciones normales de operación. Estos no representan ningún daño potencial al organismo incluso durante largos tiempo de exposición directa a los ojos y/o la piel. Este tipo de rayos láser deben ser etiquetados para indicar su clase como lo muestra la Figura 4. Sin embargo, no existen requisitos especiales de seguridad para su uso (Rockwell Laser Industries, 2001).

Figura 4: Indicador de rayos láser Clase 1

Fuente: **(Wikipedia, 2012)**

b. Clase 1M Este tipo de rayos láser no pueden producir niveles de radiación dañinos bajo condiciones normales de operación a menos que sean utilizados con algún tipo de instrumento óptico, como una lupa o un telescopio. Los dispositivos que utilicen rayos láser de este tipo deberán de ser etiquetados para indicar su clase. El único requisito de seguridad para el uso de este tipo de rayos es prevenir la exposición a los mismos a través de instrumentos ópticos. Además, los rayos láser Clase 1M poseen longitudes de onda entre los 302.5 y 4000 nm (Rockwell Laser Industries, 2001).

c. Clase 2 Son dispositivos de baja potencia (< 1mW) con un haz de luz láser en el rango visible (400 – 700 nm) que podrían causar daño a la vista de una persona si son vistos directamente durante períodos de tiempo mayores a los 0.25 segundos. Sin embargo, el reflejo de parpadeo de los ojos limita la exposición a los mismos a no más de 0.25 segundos (Rockwell Laser Industries, 2001).

La supresión intencional del parpadeo de los ojos podría ocasionar una lesión ocular. La mayoría de punteros láser y algunos instrumentos de medición utilizan este tipo de rayos láser. Estos dispositivos deben ser etiquetados como lo muestra la Figura 5 (Lasermet, Ltd, 2012).

Figura 5: Indicador de rayos láser Clase 2

Fuente: (Wikipedia, 2012)

d. Clase 2M Este tipo de rayos láser son visibles y, al igual que los rayos Clase 2, no presentan ningún riesgo de lesión al ser vistos accidentalmente mientras ocurra el reflejo del parpadeo de los ojos para evitar tiempos de exposición superiores a los 0.25 segundos. Sin embargo, podrían ocasionar lesiones oculares al ser vistos, incluso accidentalmente, utilizando instrumentos ópticos al igual que los rayos Clase 1M (Lasermet, Ltd, 2012).

e. Clase 3R La radiación emitida por este tipo de rayos láser es considerada de bajo riesgo pero potencialmente peligrosa. Su longitud de onda está comprendida entre los 302.5 y 106 nm; el riesgo de producir una lesión ocular existe pero es menor que en el caso de los rayos Clase 3B.

No presentan peligro de provocar un incendio y podrían ocasionar daños severos al utilizar instrumentos ópticos para verlos (Rockwell Laser Industries, 2001). Los dispositivos que utilicen este tipo de rayos deben ser etiquetados como lo muestra la Figura 6.

Figura 6: Indicador de rayos láser Clase 3R

Fuente: (Wikipedia, 2012)

f. Clase 3B La exposición visual directa y radiación emitida por este tipo de rayos láser son peligrosas para los ojos y la piel. Sin embargo, las reflexiones de la radiación de este tipo de rayos en papel, paredes o superficies mate no provocan ningún daño (Rockwell Laser Industries, 2001).

g. Clase 4 Este clase incluye a todos los rayos láser que exceden las limitantes de la Clase 3B. La exposición directa e indirecta a rayos láser de este tipo puede quemar la piel y causar lesiones oculares severas permanentes. Incluso la reflexión de la radiación de estos rayos puede causar lesiones permanentes en los ojos y la piel. Además, este tipo de rayos láser podrían encender materiales combustibles y por lo tanto representan un riesgo alto de incendio (Lasermet, Ltd, 2012).

Los dispositivos que utilicen rayos Clase 4 deben poseer un interruptor con llave y un bloqueo de seguridad reducir el riesgos de lesiones permanentes y/o incendios. Este tipo de rayos láser son utilizados comúnmente en aplicaciones industriales, militares, médicas y científicas (Lasermet, Ltd, 2012). Los dispositivos que utilicen rayos Clase 4 deben estar etiquetados como lo muestra la Figura 7.

Figura 7: Indicador de rayos láser Clase 4

Fuente: (Wikipedia, 2012)

C. Caracterización de un instrumento de medición

1. Caracterización estática de un instrumento de medición Las características estáticas de un instrumento de medición son todas aquellas propiedades del instrumento luego de que todos los efectos transitorios del sistema han alcanzado su estado estable (Wright State University, 2010). La Tabla 2 muestra las características estáticas más importantes de un instrumento de medición.

Tabla 2: Características estáticas de un instrumento de medición

Característica	Descripción
Tipo de salida	Indica el tipo de salida que ofrece el instrumento de medición; esta puede ser analógica o digital (Esquit, 2010).
Tipo del instrumento	**Indicación:** Se refiere a la capacidad del instrumento de medición de indicar el valor de la medición de forma visual o auditiva. **Transmisión:** Se refiere a la capacidad del instrumento de medición de transmitir el valor de la medición hacia una unidad de procesamiento centralizado (Esquit, 2010).
Exactitud	Es una medida de la cercanía del valor medido con respecto al valor real. Se representa por medio de un porcentaje (%) de la escala completa (Esquit, 2010).
Precisión	Representa la capacidad del instrumento para generar la misma medición cuando el valor de la variable es medido en repetidas ocasiones (Esquit, 2010).
Repetitividad	Expresa la capacidad del instrumento para ofrecer la misma medición cuando el valor de la variable es medido en repetidas ocasiones a través del tiempo y bajo condiciones ambientales similares (Esquit, 2010).
Reproductibilidad	Expresa la capacidad del instrumento para ofrecer la misma medición cuando el valor de la variable medida es el mismo pero las condiciones ambientales son distintas (Esquit, 2010).

Continuación Tabla 2

Característica	Descripción
Rango	Expresa los valores mínimos y máximos que el instrumento puede medir confiablemente (Esquit, 2010).
Umbral	Representa el mínimo valor de la variable medida que puede ser detectado por el instrumento (Esquit, 2010).
Espacio muerto	Se refiere al rango de la variable de entrada para el cual el instrumento de medición no responde (Esquit, 2010).
Resolución	Es el cambio mínimo en la variable de entrada que puede ser medido por el instrumento (Esquit, 2010).

Fuente: (Esquit, 2010)

2. Caracterización dinámica de un instrumento de medición Las características dinámicas de un instrumento de medición son aquellas propiedades de la respuesta transitoria al estímulo del instrumento (Wright State University, 2010). La Tabla 3 muestra las características dinámicas más importantes de un instrumento de medición.

Tabla 3: Características dinámicas de un instrumento de medición

Característica	Descripción
Tiempo de respuesta	Indica el tiempo que tarda el sensor en responder a un cambio abrupto en la variable de entrada (Esquit, 2010).
Frecuencia de muestreo	Es el número de mediciones por segundo que realiza el instrumento de medición (Esquit, 2010).

Fuente: (Esquit, 2010)

D. Estándar USB

1. Antecedentes El Bus Serial de Comunicación (USB por sus siglas en inglés) es un estándar de comunicación serial desarrollado en 1995 por Compaq, Intel, Microsoft, NEC, Hewlett-Packard, Lucent y Philips. El USB fue desarrollado como un nuevo medio a través del cual poder conectar dispositivos periféricos a una computadora y, eventualmente, remplazar a los puertos seriales y paralelos. Los objetivos principales del estándar eran la facilidad de uso y el bajo costo (USB made simple, 2010).

Uno de los retos de los desarrolladores del USB en aquel entonces fue lograr que los usuarios no necesitaran tener conocimiento especializado para poder instalar un nuevo dispositivo periférico en la computadora. Además, se requería que la computadora pudiese diferenciar los dispositivos conectados al bus y así ejecutar el manejador o driver adecuado automáticamente (USB made simple, 2010).

En la actualidad, el estándar USB permite comunicar una amplia gama de periféricos con una computadora; desde dispositivos de uso general, como apuntadores, cámaras fotográficas, impresoras o teclados, hasta dispositivos para aplicaciones específicas, como sensores industriales, teléfonos celulares, módulos GSM, adaptadores de red, discos duros externos o consolas de videojuegos. Por esta razón el USB ha reemplazado a la mayoría de protocolos de comunicación serial existentes (Otten K. , 2008). El logotipo oficial del estándar USB se muestra en la Figura 8.

Figura 8: Símbolo del estándar USB

Fuente: (Wikipedia, 2012)

2. Introducción al estándar USB El bus de comunicación USB es controlado por un anfitrión o controlador USB (*USB host* en inglés). Estese encarga de controlar todas las transacciones en el bus de comunicación y administrar el ancho de banda del mismo. La especificación USB no permite que exista más de un controlador USB en el mismo bus de comunicación (Peacock, 2010).

Bajo el estándar USB los dispositivos periféricos no pueden comunicarse entre sí; pueden comunicarse únicamente con el controlador USB del bus; los anfitriones USB más comunes son las computadoras. Un sistema típico USB está compuesto por un controlador USB y uno o más dispositivos periféricos. Los dispositivos periféricos responden únicamente a peticiones del controlador; estos no son capaces de iniciar una transferencia de información (Otten K. , 2008).

El controlador USB se encarga de iniciar la comunicación a través del bus. Es decir, un dispositivo periférico puede enviar información hacia el controlador USB solamente cuando este último se la solicite; además, el periférico debe ser capaz de recibir la información enviada por el controlador (Otten K. , 2008).

3. Teoría de operación Cuando algún dispositivo periférico es conectado al bus de comunicación ocurre lo siguiente:

El controlador detecta la conexión del periférico y lo interroga para obtener información acerca del mismo. Luego, el controlador ejecuta automáticamente el "driver" o controlador apropiado para esa clase de dispositivo USB e informa al usuario que el periférico está listo para ser utilizado. Dicho proceso recibe el nombre de "Enumeración". Posteriormente, cuando el dispositivo periférico es desconectado del bus, el controlador detectará la desconexión y automáticamente terminará la ejecución del "driver" o controlador asociado (Peacock, 2010).

4. Especificaciones

a. Velocidades de transmisión El estándar USB en su versión 1.1 soporta dos velocidades de transmisión: un modo de baja velocidad (*low speed* en inglés) de 1.5 Mbits/s y otro de velocidad completa (*full speed* en inglés) de 12 Mbits/s. Además, el USB 2.0 soporta una velocidad de transmisión de 480 Mbits/s conocida como alta velocidad (*high speed* en inglés). Por último, el USB 3.0 soporta una tasa de transmisión de 4.8 Gbits/s conocida como super alta velocidad (*super - high speed* en inglés). Entonces, el estándar USB soporta cuatro velocidades de transmisión (Peacock, 2010):

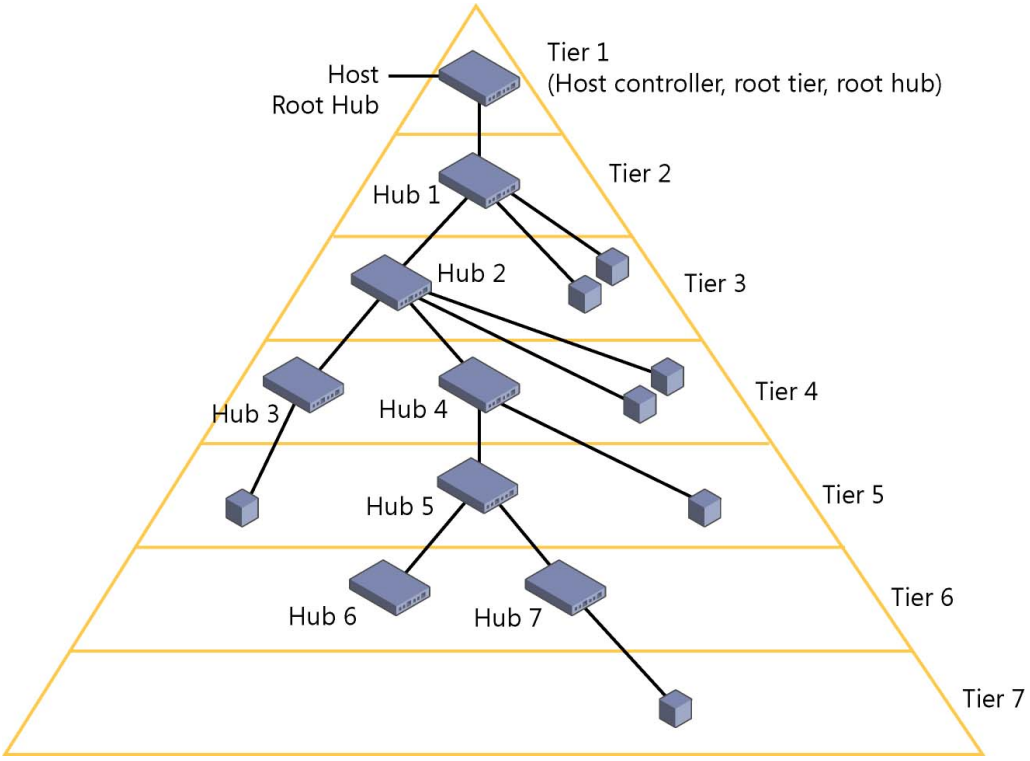
- Low Speed (LS) – 1.5 Mbits/s (187.5 KB/s)
- Full speed (FS) – 12 Mbits/s (1.5 MB/s)
- High speed (HS) – 480 Mbits/s (60 MB/s)
- Super High speed (SHS) – 4.8 Gbits/s (600 MB/s)

b. Topología USB El estándar USB está basado en una topología de "estrella de niveles" en donde existe un solo anfitrión o controlador USB y hasta 126 dispositivos periféricos conectados al bus de comunicación. Esta limitante se debe a que, según la especificación USB, el campo de dirección en un paquete de información tiene 7 bits de longitud y la dirección 0x00 está reservada. Es decir, el Host USB puede asignar a los periféricos conectados al bus hasta (2^7) - 1 = 127 direcciones comprendidas entre 1 (0x01) y 127 (0x7F) (USB made simple, 2010).

Los controladores Host USB, como las computadoras, poseen un concentrador o "root hub" integrado que provee un número inicial de puntos de conexión al bus o "puertos". La arquitectura del "root hub" está integrada en las tarjetas madre de todas las computadoras y representa el primer nivel de la topología del bus de comunicación USB (USB made simple, 2010).

Sin embargo, la topología del bus USB puede expandirse conectando un nuevo "Hub 1" en algún puerto disponible del "root hub" para incrementar el número de niveles. Luego, es posible conectar otro "Hub 2" en algún puerto disponible del "Hub 1" y así sucesivamente. De esta forma puede incrementarse el número de niveles de la topología hasta un máximo de 7 (véase Figura 9) (USB made simple, 2010).

Figura 9: Topología del bus USB



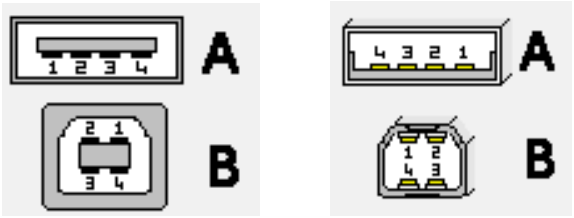
Fuente: (Microsoft, 2005)

Según la especificación USB, el "root hub" ocupa el primer nivel de la topología y está implementado en el Anfitrión USB del bus. Además, el bus soporta la conexión de hasta 5 "non-root hubs" en niveles posteriores (nivel 2 – nivel 6) para ampliar la cantidad de puertos. Sin embargo, el último nivel de la topología (nivel 7) soporta la conexión de un único dispositivo periférico que no sea un "hub" (Microsoft, 2005).

c. Conectores y patillaje El estándar USB soporta varios tipos de conectores dependiendo de la versión del mismo. Sin embargo, se utilizan distintos tipos de conectores en los dispositivos periféricos y en los controladores Host USB para asegurar su conexión correcta al bus de comunicación y evitar violaciones a la topología de la especificación USB. El conector ubicado en el *Host USB* o en los dispositivos periféricos recibe el nombre de receptáculo ("receptacle" o "jack" en inglés) o hembra y los conectores añadidos a las puntas de los cables USB reciben el nombre de "plugs" o machos (PINOUTS.RU, 2011).

En las versiones originales del estándar USB (1.0/1.1) se detallan los receptáculos y "plugs" tipo Standard-A y Standard-B (véase Figura 10). Este tipo de conectores fueron diseñados según la topología del estándar USB: Se utilizan conectores tipo Standard-A en los Host USB que entregan potencia al bus y conectores tipo Standard-B en los dispositivos periféricos que demandan dicha potencia del bus (PINOUTS.RU, 2011).

Figura 10: Receptáculos (izq.) y plugs (der.) tipo Standard A y B



Fuente: (PINOUTS.RU, 2011)

La Tabla 4 muestra el patillaje y código de color del cableado para los conectores tipo Standard-A y Standard-B (PINOUTS.RU, 2011).

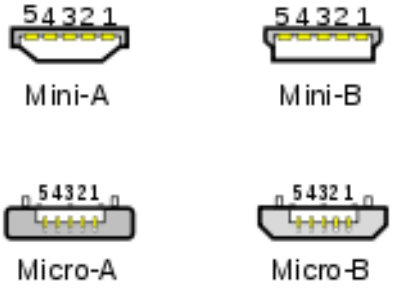
Tabla 4: Patillaje de conectores USB estándar

No. de pin	Nombre	Color de cable	Descripción
1	V_{BUS}	Rojo	Alimentación del bus (5 VDC)
2	D-	Blanco	Datos -
3	D+	Verde	Datos +
4	GND	Negro	Tierra

Fuente: (PINOUTS.RU, 2011)

El estándar USB en su versión 2.0 añade la descripción de receptáculos y *plugs* tipo Mini-A, Mini-B, Micro-A y Micro-B (véase Figura 11). Este tipo de conectores fueron diseñados para conectar a la computadora dispositivos periféricos más pequeños como PDAs, teléfonos celulares o cámaras fotográficas digitales (Peacock, 2010).

Figura 11: Plugs tipo Mini-A/Mini-B y Micro-A/Micro-B



Fuente: (Wikipedia, 2012)

La Tabla 5 muestra el patillaje y código de color del cableado para los conectores tipo Mini-A/Mini-B y Micro-A/Micro-B (USB made simple, 2010).

Tabla 5: Patillaje de conectores USB Mini-A/Mini-B, Micro-A/Micro-B

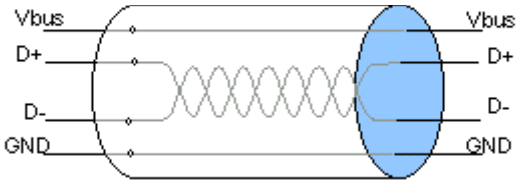
No. de pin	Nombre	Color del cable	Descripción
1	V_{BUS}	Rojo	Alimentación del bus (5 VDC)
2	D-	Blanco	Datos -
3	D+	Verde	Datos +
4	ID	-	Utilizado en los dispositivos USB On-The-Go (OTG) para determinar el rol del dispositivo.
5	GND	Negro	Tierra

Fuente: (PINOUTS.RU, 2011)

d. Cableado La especificación USB 2.0 o inferior establece el uso de un cable blindado de 4 hilos. Dos de estos, D+ y D-, forman un par trenzado y son responsables de transmitir una señal de datos diferencial (USB made simple, 2010).

Estas señales están referenciadas a la línea GND (0 VDC) del cable. Por último, la línea V_{BUS} provee una alimentación de 5 VDC que puede ser utilizada por los dispositivos periféricos conectados al bus (véase Figura 12) (USB made simple, 2010).

Figura 12: Composición de un cable USB



Fuente: (USB made simple, 2010)

e. Potencia La especificación USB 2.0 o anterior provee una fuente de alimentación de 5 VDC en una línea del bus (V_{BUS}) desde la cual los dispositivos periféricos conectados podrían ser alimentados. Dicha especificación provee una alimentación de no menos de 4.75 V y no más de 5.25 V (5 V ± 5%) en V_{BUS}. Por otro lado, el estándar USB 3.0 provee una alimentación comprendida entre 4.45 V y 5.25 V (USB made simple, 2010).

La especificación USB define una "carga unitaria" como 100 mA en el estándar USB 2.0 y 150mA en el estándar USB 3.0.El estándar USB 2.0 establece que un dispositivo puede demandar hasta 5 cargas unitarias (500 mA) del mismo puerto, mientras que el estándar USB 3.0 permite que un dispositivo demande como máximo 6 cargas unitarias (900 mA) (USB made simple, 2010).

Los dispositivos periféricos se clasifican según su consumo de potencia en dos tipos: dispositivos de "bajo consumo" y dispositivos de "alto consumo". Sin embargo, el estándar USB establece que los dispositivos periféricos, sin importar su tipo, no pueden demandar más de 1 carga unitaria (100 mA/150 mA) durante la fase de enumeración y configuración (USB made simple, 2010).

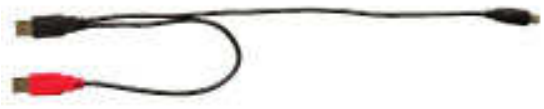
1) Dispositivos de bajo consumo Según la especificación USB 2.0, los dispositivos periféricos de bajo consumo demandan a lo sumo 1 carga unitaria (100 mA) del puerto USB al que están conectados durante la fase de enumeración y configuración e incluso después de estas (USB made simple, 2010).

2) Dispositivos de alto consumo Los dispositivos periféricos de alto consumo demandan a lo sumo 1 carga unitaria (100 mA) durante la fase de enumeración y configuración; luego de estas efectúan una transición y demandan el número máximo de cargas unitarias permitidas por el estándar. Por esta razón, los dispositivos de alto consumo podrán demandar hasta 5 cargas unitarias (500 mA) solo después de haber sido enumerados y configurados por el Host USB (USB made simple, 2010).

Algunos dispositivos, como los discos duros externos, requieren más de 500 mA para funcionar debido a que poseen componentes mecánicos internos que demandan corriente adicional (i.e. motores) y por lo tanto no pueden ser alimentados de un único puerto USB 2.0. Estos dispositivos usualmente incluyen un cable modificado en forma de 'Y' con dos *plugs* tipo Standard-A que se conectan a dos puertos independientes del Host USB (i.e. una computadora) (véase Figura 13).

Estos cables permiten que el dispositivo obtenga potencia de dos puertos USB simultáneamente (Hokuyo, 2009).

Figura 13: Cable USB modificado en forma de Y



Fuente: (Hokuyo, 2009)

5. Dispositivos USB Los dispositivos USB son cualquier tipo de dispositivo periférico que implementa una función en específico y que puede conectarse a un puerto USB. Todos los dispositivos USB contienen información de configuración acerca de sus características y los recursos requeridos para funcionar. Esta información está contenida en el "descriptor de dispositivo" y es interpretada por el Host USB durante el proceso de enumeración (Microsoft, 2005).

Al ser conectados al bus, los dispositivos USB son reconocidos, enumerados e inicializados por el Host USB. Al terminar dicho proceso estos pueden comenzar a ser utilizados sin requerir instalaciones o configuraciones adicionales por parte del usuario final (Microsoft, 2005). Esta es una de las principales ventajas del estándar USB sobre otros protocolos de comunicación más antiguos como el serial, paralelo o PS-2.

La especificación USB clasifica a los dispositivos periféricos según su función en las siguientes clases: *Communications Device Class* (CDC), *Human Interface Device Class* (HID), *Mass Storage Device Class* (MSD), *Printer Class*, *Audio Class*, *Battery Charging Class*, *Imaging Class, Physical Interface Devices (PID), Smart Card Class, IrDA Bridge*, entre otras (Microsoft, 2005). A continuación se describen brevemente las clases de dispositivos USB más comunes:

a. Communications Device Class (CDC) La *Communications Device Class* (CDC) es una clase de dispositivo de la especificación USB que agrupa a los dispositivos periféricos que transmiten grandes cantidades de datos e información hacia una computadora a través del USB. Los dispositivos CDC más comunes en el mercado son las impresoras, adaptadores de red, módems, routers, sensores industriales, entre otros (Microchip Technology, 2008).

b. Human Interface Device Class (HID) La *Human Interface Device Class* (HID) es una clase de dispositivo de la especificación USB que agrupa a los dispositivos periféricos que se conectan usualmente a una computadora. Los dispositivos HID más comunes son teclados, apuntadores o mouse, UPS, teclados numéricos, controladores de juego o "joysticks" (Microchip Technology, 2008).

c. Mass Storage Device Class (MSD) La *Mass Storage Device Class* es una clase de dispositivo de la especificación USB que agrupa la mayoría de dispositivos periféricos de almacenamiento masivo. Esta define una interfaz que permite leer, escribir y borrar sectores de datos e incluso realizar particiones en los dispositivos de almacenamiento desde un Host USB como una computadora (Microchip Technology, 2008).

Algunos de los dispositivos periféricos que utilizan este estándar son: discos duros externos magnéticos, lectores ópticos de CD/DVD, memorias flash portables, cámaras digitales, reproductores de música, PDAs, teléfonos celulares, entre otros (Microchip Technology, 2008).

6. Anfitriones USB El anfitrión o Host USB es el encargado de controlar el tráfico de información en el bus de comunicación. Además, realiza las siguientes tareas (Microchip Technology, 2008):

- Provee alimentación al bus e inicia todas las transferencias de datos
- Administra el flujo de información y maneja la detección de errores
- Detecta la conexión/desconexión de los dispositivos periféricos
- Enumera a los periféricos conectados al bus y ejecuta el controlador o "driver" adecuado según la clase de dispositivo USB

Como se mencionó arriba, los Host USB se encargan de proveer la alimentación al bus. Estos deben proveer el máximo número de cargas unitarias permitidas por el estándar (5 cargas unitarias / 500 mA en USB 2.0) en cada uno de los puertos del "root hub" (USB made simple, 2010).

Los anfitriones USB pueden clasificarse según el tipo de microarquitectura en donde estén implementados y las funciones que desempeñan; la clasificación se detalla a continuación.

a. Anfitrión USB de propósito general Los anfitriones USB de propósito general (Full USB Host, en inglés) usualmente están implementados en microarquitecturas con generosos recursos de procesamiento y memoria como la de una computadora. Los conectores utilizados en este clase de Host USB son receptáculos tipo Standard-A (véase Figura 10) integrados en el "root hub" de la tarjeta madre de la computadora (Otten K. , 2008).

Esta clase de anfitrión o Host debe soportar cualquier clase de dispositivo USB (i.e. *HID, CDC, MSD, PID, Printer Class, Audio Class*). Para ello es necesario, en algunos casos particulares, instalar controladores o "drivers" especiales en el Host antes de conectar el dispositivo periférico al puerto. Además, deben soportar la conexión de concentradores de puertos o "hubs" al bus (Otten K. , 2008).

b. Anfitrión USB dedicado Los anfitriones USB dedicados (USB Embedded Host, en inglés) usualmente están implementados en dispositivos de doble rol con funcionalidad On-The-Go (OTG) (véase *"Dispositivos de doble rol"*, pág. 41) que poseen microarquitecturas con recursos limitados de procesamiento y memoria como la de un microcontrolador, teléfonos celulares, cámaras digitales o un circuito integrado dedicado (Otten K. , 2008).

Esta clase de anfitrión dedicado soporta únicamente dispositivos periféricos específicos y/o clases de dispositivos específicos definidos en la *Targeted Peripheral List* (TPL). Por esta razón, soporta solo aquellos tipos de transferencias requeridos por los dispositivos conectados al bus. Además, el soporte de concentradores de puertos o "hubs" es opcional y los requerimientos de potencia son limitados (Otten K. , 2008).

1) *Targeted Peripheral List* (TPL) La TPL es una lista que detalla los dispositivos periféricos soportados por un anfitrión USB dedicado. Cada dispositivo periférico de la lista es identificado según su fabricante, clase de dispositivo USB y número de modelo (USB.org, 2001).

Los tipos de transferencias soportados además de las transferencias de control dependen de las características y la clase de dispositivo USB de los periféricos contenidos en esta lista (USB.org, 2001). La Tabla6 resume las principales diferencias entre ambos tipos de anfitriones USB.

Tabla6: Anfitrión USB vs anfitrión USB dedicado

Anfitrión USB	Anfitrión USB dedicado
• Soporta todas las clases de dispositivos USB.	• Soporta únicamente dispositivos específicos y/o clases de dispositivos específicas definidos en la TPL.
• Soporta todos los tipos de transferencias principales: Control, Masivas, Isócronas, Interrupción. (véase "Transferencias", pág. 34)	• Soporta las transferencias de Control y opcionalmente puede soportar uno de los siguientes tipos de transferencias: Masivas, Isócronas o de Interrupción según la TPL.
• Soporta la conexión de "hubs".	• No soporta la conexión de "hubs".
• Pueden proveer 500 mA en cada uno de sus puertos.	• Debe ser capaz de proveer por lo menos 8 mA en el V_{BUS} del puerto USB.

Fuente: (Microchip Technology, 2008)

La Tabla 7 describe los tipos de transferencias asociados a cada clase de dispositivo periférico USB según sus características.

Tabla 7: Tipos de transferencias USB

Clase de dispositivo USB	Tipos de transferencias implementados
Communications Device Class	Control y masivas
Human Interface Class	Control y de interrupción
Mass Storage Device Class	Control y masivas
Audio Class	Control e isócronas

Fuente: (Microchip Technology, 2008)

7. Transferencia de información Según la especificación USB, cuando el Host USB transmite un paquete de información este es enviado a todos los dispositivos periféricos conectados a un puerto habilitado del bus. Sin embargo, solo el dispositivo direccionado acepta el paquete; el resto de dispositivos lo reciben pero lo rechazan porque su dirección no coincide con la del paquete.

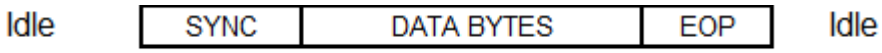
Por otro lado, el estándar USB permite que solo un dispositivo transmita información hacia el Host USB simultáneamente en respuesta a una petición de este último (USB made simple, 2010).

a. Paquetes De acuerdo a la especificación USB, los paquetes son el elemento más pequeño de transmisión de información. Cada paquete contiene un número específico de bytes que son transmitidos a la velocidad de transmisión actual del bus comenzando por el bit menos significativo (*Lowest-Significant Bit First*, en inglés). Mientras no hay tráfico de paquetes en el bus de comunicación, este se encuentra en estado de reposo o "idle" (USB made simple, 2010).

La sincronización, relleno de bits e identificación del inicio/fin de los paquetes de datos es llevada a cabo por un módulo de circuitería electrónica dedicada llamado *"Motor de interfaz serial"* (SIE por sus siglas en inglés). El SIE implementa todas las tareas críticas de tiempo de la especificación USB. Además, es capaz de identificar el número de bytes contenidos en un paquete de datos (USB made simple, 2010).

La estructura básica de un paquete está formada por 3 campos de información (véase Figura 14). El primero es un campo de sincronización (Sync), el segundo es el campo de datos (DATA BYTES); la información contenida en este campo depende del tipo de paquete (véase Tabla 8). Por último, el campo EOP indica el final del paquete (USB made simple, 2010).

Figura 14: Estructura de un paquete de datos



Fuente: (Peacock, 2010)

Tabla 8: Campos de un paquete

Campo	Longitud	Descripción
Sync	8 bits (LS, FS) 32 bits (HS)	Es utilizado para sincronizar el reloj del receptor del paquete con el del transmisor. Esta sincronización es llevada a cabo por el SIE.
PID	8 bits	Representa el *"Packet Identifier"* y es utilizado para identificar el tipo de paquete que está siendo enviado.
ADDR	7 bits	Representa la dirección del dispositivo periférico al cual está dirigido el paquete. Esta está comprendida entre 1 (0x01) y 126 (0x0F).
ENDP	4 bits	Indica el número de Endpoint al cual está dirigido el paquete.
CRC	5 bits (token) 16 bits (data)	Representa el *Cyclic Redundance Check* que permite identificar errores en la transmisión del paquete. Su longitud depende del tipo de paquete (*token packet*/*data packet*).
EOP	3 bits	Representa el final del paquete y es identificado por el SIE.

Fuente: (Peacock, 2010)

Según la especificación USB, el *Serial Interface Engine* (SIE) debe reconocer rápidamente el *Packet Identifier* (PID) de cada paquete y por esta razón no se utiliza el *Cyclic Redundance Check* (CRC) para verificar que este se haya recibido correctamente. Sin embargo, el byte PID posee su propia verificación de recepción: los 4 bits del PID se complementan y se concatenan al PID original para formar el campo de 8 bits contenido en el paquete (véase Figura 15). La especificación USB soporta 4 tipos de paquetes; estos se describen a continuación (Peacock, 2010).

Este byte es transmitido según el orden de transmisión establecido por la especificación USB: *Lowest-Significant Bit First*, en inglés (Peacock, 2010).

Figura 15: Byte de Packet ID

Fuente: (USB made simple, 2010)

Tabla 9: Valores del PID

Tipo de paquete	Nombre del paquete	PID<3:0>
Token	OUT Token	0001b
	IN Token	1001b
	SOF Token	0101b
	SETUP Token	1101b
Data	DATA0	0011b
	DATA1	1011b
	DATA2	0111b
	MDATA	1111b
Handshake	ACK Handshake	0010b
	NAK Handshake	1010b
	STALL Handshake	1110b
	NYET	0110b
Special	PRE	1100b
	ERR	1100b
	SPLIT	1000b
	PING	0100b
	Reservado	0000b

Fuente: (Peacock, 2010)

1) Paquetes "token" Este tipo de paquetes son transmitidos durante la etapa de configuración (SETUP stage) de una transacción y contienen información acerca del *endpoint* direccionado y el propósito de la transacción. Los tipos de paquetes "token" que define el estándar USB se muestran en la Tabla 10 (Peacock, 2010).

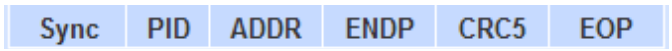
Tabla 10: Tipos de paquetes "Token"

Tipo	Descripción
IN	Informan al dispositivo periférico que el Host requiere recibir información.
OUT	Informan al dispositivo periférico que el Host requiere enviarle información.
SETUP	Es utilizado para iniciar una transferencia de control.

Fuente: (Peacock, 2010)

Para los paquetes SOF véase Paquetes "Start-of-Frame" pág. 31 . Los paquetes tipo "token" deben cumplir con el formato mostrado en la Figura 16.

Figura 16: Formato de los paquetes "Token"



Fuente: (Peacock, 2010)

2) Paquetes de datos Este tipo de paquetes son capaces de transmitir hasta 1024 bytes de datos y son utilizados durante la etapa de datos (DATA stage) de una transacción (véase Tabla 11) (Peacock, 2010).

Tabla 11: Tipos de paquetes de datos

Tipo	Descripción
DATA0 **DATA1**	Pueden transmitir 1024 bytes de datos según la velocidad de transmisión del dispositivo periférico. - Velocidad baja – 8 bytes - Velocidad completa – 1023 bytes - Velocidad alta – 1024 bytes
DATA2 **MDATA**	Pueden transmitir hasta 1024 bytes de datos y están definidos únicamente en el modo de alta velocidad.

Fuente: (Peacock, 2010)

Los paquetes de datos deben cumplir con el formato mostrado en la Figura 17.

Figura 17: Formato de los paquetes de datos

Fuente: (Peacock, 2010)

3) Paquetes "handshake" Este tipo de paquetes se utilizan en la tapa de estado (STATUS stage) de una transacción e indican el estado de la transacción actual. Los tipos de paquetes "handshake" que define el estándar USB se muestran en la Tabla 12 (Peacock, 2010).

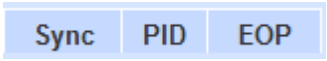
Tabla 12: Tipos de paquetes "handshake"

ACK	Reconocimiento de que el paquete anterior se ha recibido satisfactoriamente.
NAK	Indica que el dispositivo receptor no puede recibir datos o que el emisor no puede enviar datos.
STALL	Indica que el dispositivo periférico requiere intervención del Host.

Fuente: (Peacock, 2010)

Los paquetes "handshake" deben cumplir con el formato mostrado en la Figura 18.

Figura 18: Formato de los paquetes "handshake"



Fuente: (Peacock, 2010)

4) Paquetes "Start-of-Frame" Los paquetes "Start-of-Frame" (SOF) contienen en un número de "frame" de 11 bits de longitud que es enviado por el Host USB cada vez que inicia un nuevo frame. El "frame" es utilizado como un marco de tiempo dentro del cual se programan las transferencias de datos requeridas (USB made simple, 2010).

La duración de un "frame" en dispositivos de baja velocidad y velocidad completa es de 1 ms mientras que en los dispositivos de alta velocidad es de 125 μs. Los paquetes SOF deben cumplir con el formato mostrado en la Figura 19 (USB made simple, 2010).

Figura 19: Formato de los paquetes "Start-of-Frame"

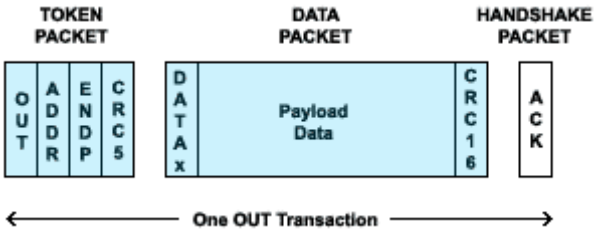
Sync	PID	Frame Number	CRC5	EOP

Fuente: (Peacock, 2010)

b. Transacciones Según el estándar USB, una transacción es una secuencia de tres paquetes que realizan una transferencia de información. Sin embargo, en algunos casos, como en las transacciones utilizadas en transferencias "isochronous", el último paquete de "handshake" es omitido porque no se requiere la detección de errores (USB made simple, 2010).

1) Transacción de salida (OUT) Una transacción de salida está formada por 3 paquetes secuenciales que realizan una transferencia de información hacia algún dispositivo periférico (USB made simple, 2010). El formato de una transacción de salida se muestra en la Figura 20.

Figura 20: Formato de una transacción de salida

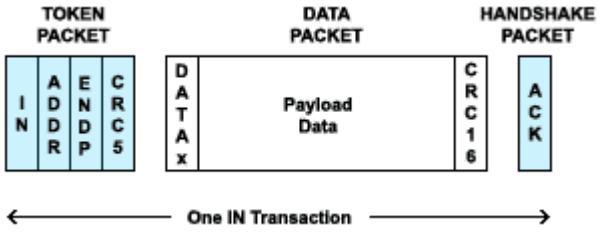


Fuente: (USB made simple, 2010)

Primero, el Host USB transmite el paquete "token" para indicar que requiere enviar datos al dispositivo en la dirección ADDR en el *endpoint* ENDP. Luego, el Host transmite el paquete que contiene los datos y el periférico responde con un paquete "handshake" indicando que el paquete de datos ha sido recibido satisfactoriamente (ACK). Sin embargo, si la transacción de salida fuese utilizada en una transferencia "isochronous" el dispositivo periférico omitiría el paquete "handshake" (USB made simple, 2010).

2) Transacción de entrada (IN) Una transacción de entrada está formada por 3 paquetes secuenciales que transmiten información desde algún dispositivo periférico hacia el Host USB (USB made simple, 2010). El formato de una transacción de entrada se muestra en la Figura 21.

Figura 21: Formato de una transacción de entrada

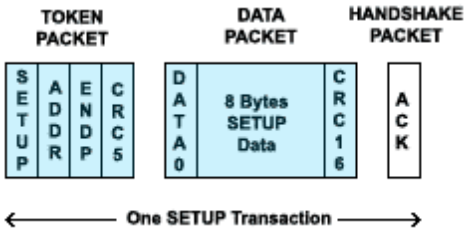


Fuente: (USB made simple, 2010)

Primero, el Host USB transmite el paquete "token" para indicar que requiere recibir datos del dispositivo en la dirección ADDR en el *endpoint* ENDP. Luego, el dispositivo direccionado transmite el paquete que contiene los datos y el Host responde con un paquete "handshake" indicando que el paquete de datos ha sido recibido satisfactoriamente (ACK). Sin embargo, si la transacción de entrada fuese utilizada en una transferencia "isochronous" el Host USB omitiría el paquete "handshake" (USB made simple, 2010).

3) Transacción SETUP Una transacción de configuración está formada por 3 paquetes secuenciales que transmiten información de configuración hacia algún dispositivo periférico (USB made simple, 2010). El formato de una transacción de configuración se muestra en la Figura 22. Además, este tipo de transacción es utilizada para iniciar una transferencia de control (USB made simple, 2010).

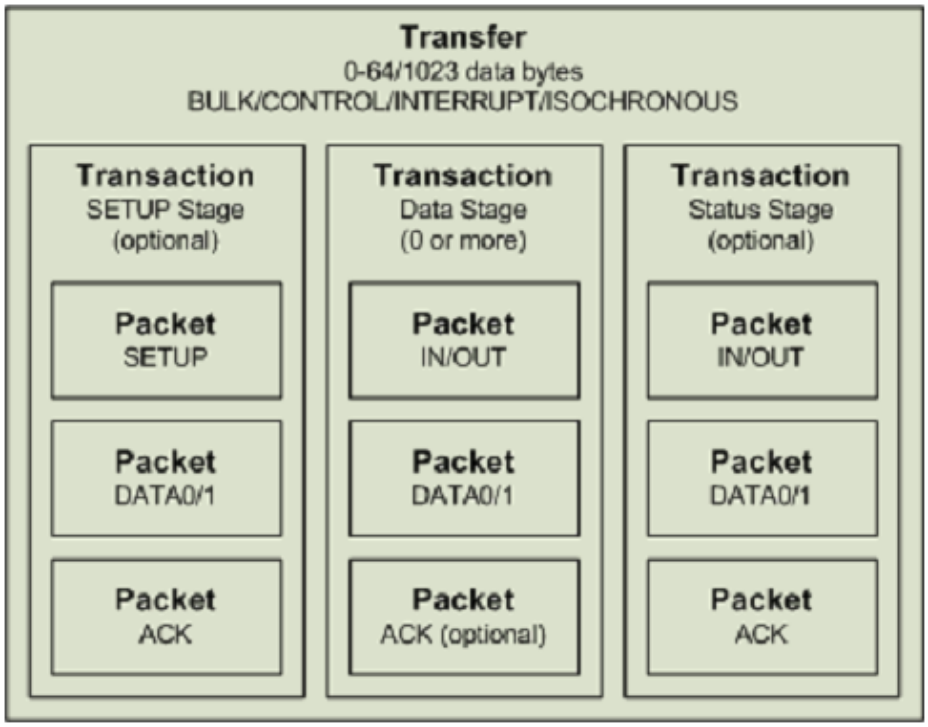
Figura 22: Formato de una transacción de configuración



Fuente: (USB made simple, 2010)

c. Transferencias Según la especificación USB, las transferencias están formadas por múltiples transacciones y permiten transmitir información a través del bus de comunicación. Además, cada una de estas transacciones está formada por múltiples paquetes secuenciales (USB made simple, 2010) (véase Figura 23).

Figura 23: Formato genérico de una transferencia USB



Fuente: (Otten K. , 2008)

Existen 4 tipos de transferencias que se diferencian según su propósito y sus características (USB made simple, 2010).

Tabla 13: Longitud máxima de los paquetes de datos

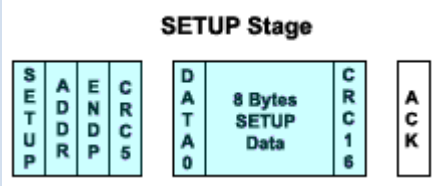
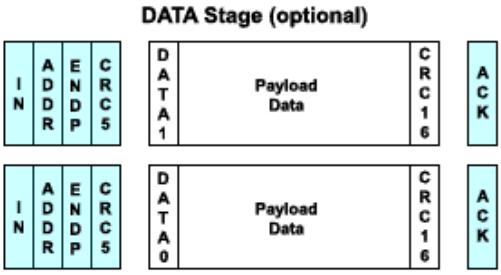
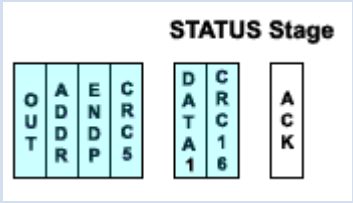
Tipo de transferencia	Longitud máxima de los paquetes de datos
Control	Low Speed(LS) - 8 bytes Full Speed (FS) – 8, 16, 32, 64 bytes High Speed (HS) - 64 bytes
Bulk	LowSpeed (LS) – no soporta el tipo de transferencia Full Speed (FS) – 8, 16, 32, 64 bytes High Speed (HS) - 512 bytes
Interrupt	Low Speed (LS) –hasta8 bytes Full Speed (FS) – hasta 64 bytes High Speed (HS) –hasta 1024 bytes
Isochronous	LowSpeed (LS) – no soporta el tipo de transferencia Full Speed (FS) –hasta 1023 bytes High Speed (HS) –hasta 1024 bytes

Fuente: (USB made simple, 2010)

1) Transferencia de control Las transferencias de control (*control transfer*s, en inglés) son transferencias bi-direccionales utilizadas por el Host USB durante el proceso de configuración y enumeración de los dispositivos periféricos (Peacock, 2010). Por esta razón, todos los Host USB deben soportar este tipo de transferencias. Sus principales características según (Peacock, 2010) son:

- Están compuestas por3 etapas (véase Tabla 14) y 2 o más transacciones
- Utilizan los 2 tipos de *endpoint*: IN/OUT
- Utilizan el *endpoint* 0 por defecto
- Son utilizadas para obtener los descriptores de dispositivo (*device descriptors*)

Tabla 14: Etapas de una transferencia de control

Etapa	Descripción	Imagen
SETUP	El Host emite una transacción SETUP (8 bytes) que contiene la solicitud del descriptor de dispositivo (*device descriptor request*) (USB made simple, 2010).	
DATA	Esta etapa es opcional y contiene las transacciones necesarias para que el dispositivo direccionado en la etapa anterior transmita su descriptor de dispositivo (*device descriptor)* (12 bytes) hacia el Host (USB made simple, 2010).	
STATUS	El Host inicia una transacción que contiene un paquete de datos DATA1 vacío para indicar que las transacciones anteriores fueron exitosas (USB made simple, 2010).	

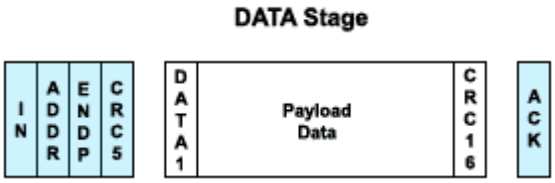
Fuente: (USB made simple, 2010)

2) Transferencia de interrupción Las transferencias de interrupción (*interrupt transfer*s, en inglés) son utilizadas para monitorear cambios de estado en un dispositivo periférico; por ejemplo, un teclado o un *mouse* (Peacock, 2010). Sus principales características según (Peacock, 2010) son:

- Están compuestas por una etapa de datos que contiene una transacción de entrada/salida (véase Figura 27)
- Proveen detección y corrección de errores utilizando un *Cyclic Redundance Check* (CRC) de 16 bits
- Poseen un ancho de banda reservado
- Transmisión de datos a una tasa de transferencia constante

El formato de una transferencia de interrupción se muestra en la Figura 24.

Figura 24: Formato de transferencias de interrupción



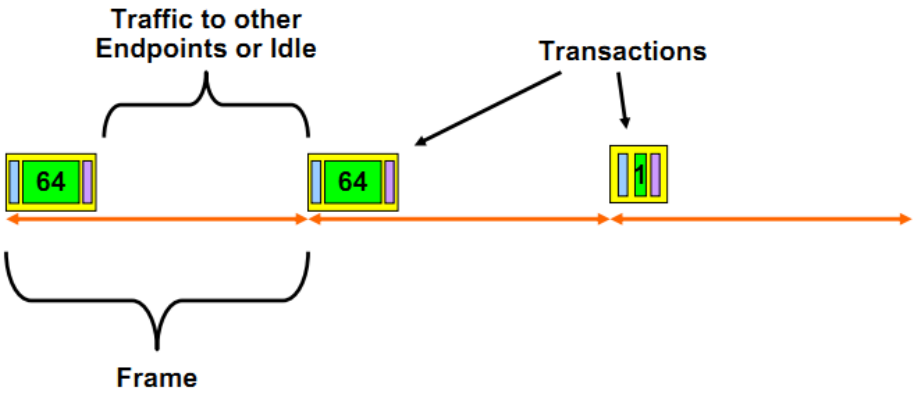
Fuente: (USB made simple, 2010)

Cuando los dispositivos periféricos cambian su estado colocan una solicitud de interrupción en la cola para indicar que requieren transmitir información urgente hacia el Host. Sin embargo, según la especificación USB, deben esperar a que el Host los consulte enviando un paquete de entrada (*IN token*) para poder transmitir la información urgente en respuesta a dicho paquete (véase Figura 24) (Peacock, 2010).

El Host consulta al dispositivo acerca de solicitudes de interrupción cada cierto intervalo de tiempo; este intervalo está definido en el *endpoint descriptor* y es expresado en números de *frames* (Peacock, 2010).

Las transferencias de interrupción (*interrupt transfers*) tienen la prioridad más alta y reservan una porción del ancho de banda del bus. Por esta razón, no existen retrasos en la transmisión y la tasa de transferencia es constante (Peacock, 2010) (véase Figura 25).

Figura 25: Ancho de banda en transferencias de interrupción



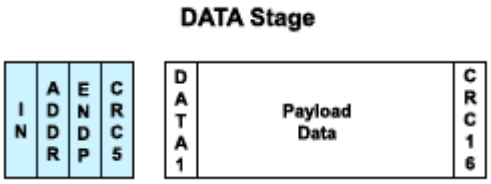
Fuente: (Colombo & Cecic, 2011)

3) Transferencias isócronas Las transferencias isócronas (*isochronous transfer*s, en inglés) son transferencias periódicas utilizadas para transmitir continuamente información sensible al tiempo; por ejemplo, pistas de audio o video (Peacock, 2010). Sus principales características según (Peacock, 2010) son:

- Están compuestas por una etapa de datos que contiene una transacción de entrada/salida dependiendo del tipo de *endpoint* (véase Figura 26)
- Proveen detección y corrección de errores utilizando un *Cyclic Redundance Check* (CRC) pero no soportan la re-transmisión o garantía de recepción
- Poseen un ancho de banda constante
- No poseen un paquete de "handshake" al final de la transacción

El formato de una transferencia isócrona se muestra en la Figura 26.

Figura 26: Formato de transferencias isócronas



Fuente: (USB made simple, 2010)

El propósito de las transferencias isócronas son aplicaciones, como la transmisión de pistas de audio y video, donde es imprescindible mantener el flujo de datos constante para evitar problemas de sincronización o porciones erráticas de audio/video debido a un retraso en la transmisión (Peacock, 2010).

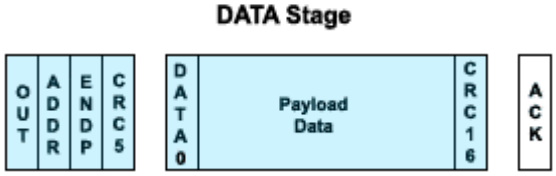
Sin embargo, por la naturaleza de la información transmitida (audio/video), es aceptable la pérdida o corrupción de algunos paquetes de datos. Además, las transferencias isócronas tienen prioridad alta y también reservan una porción del ancho de banda del bus. Por esta razón, no existen retrasos en la transmisión y su ancho de banda es constante (Peacock, 2010).

4) Transferencia masiva Las transferencias masivas (*bulk transfer*s, en inglés) son utilizadas para transmitir grandes cantidades de datos que no requieren una tasa de transferencia constante; por ejemplo, una impresión, una imagen generada por un *scanner* o las mediciones efectuadas por un sensor (Peacock, 2010). Sus principales características según (Peacock, 2010) son:

- Están compuestas por una etapa de datos que puede contener múltiples transacciones de entrada/salida dependiendo del tipo de *endpoint* (véase Figura 27)
- Proveen detección y corrección de errores utilizando un *Cyclic Redundance Check* (CRC) de 16 bits
- Re-transmisión de los datos para asegurarse que hayan sido transmitidos y recibidos sin errores
- Sin garantía de ancho de banda

El formato de una transferencia masiva de salida se muestra en la Figura 27.

Figura 27: Formato de transferencias masivas de salida

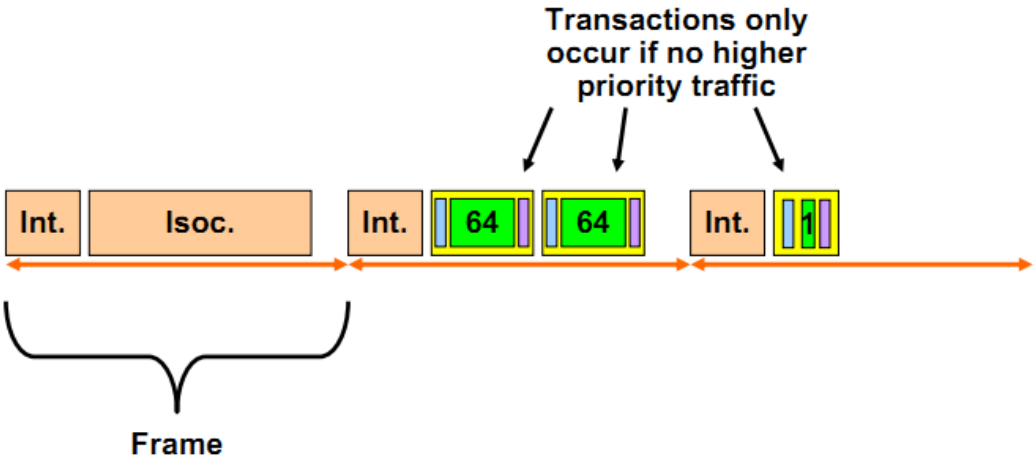


Fuente: (USB made simple, 2010)

Las transferencias masivas (*bulk transfers*) tienen la prioridad más baja y utilizan el ancho de banda "libre" en el bus después de que todas las otras transferencias han sido asignadas. Si el bus está ocupado con transferencias de control, isócronas o de interrupción, los datos masivos tendrían que esperar y "colarse" poco a poco a través del bus de comunicación (Peacock, 2010).

Por esta razón es posible que ocurran retrasos en la transmisión de los datos debido a que el ancho de banda es insuficiente en algunos *frames* (véase Figura 28). Dichos retrasos no permiten que la tasa de transferencia de datos masivos sea constante (Peacock, 2010).

Figura 28: Ancho de banda en transferencias masivas



Fuente: (Colombo & Cecic, 2011)

8. USB On-The-Go (OTG) El estándar USB es la interfaz más popular utilizada en el intercambio de información entre una computadora y dispositivos periféricos. Cada día más dispositivos periféricos portables utilizan la interfaz USB para comunicarse con una computadora. Muchos de estos dispositivos podrían extender sus funcionalidades si pudiesen comunicarse entre sí directamente sin la necesidad de una computadora que cumpla el rol de Anfitrión USB (USB.org, 2001).

a. Introducción al USB On-The-Go El estándar USB *On-The-Go* surge como una alternativa para que los dispositivos periféricos comunes puedan tomar el rol de un Host USB y se comuniquen con otros dispositivos sin la necesidad de una computadora.

Esta interfaz permite, por ejemplo, conectar memorias flash portables a un teléfono celular, imprimir fotografías desde una cámara fotográfica o administrar el contenido de un reproductor de música desde un teléfono celular, entre otras (USB.org, 2001).

El estándar USB OTG es un suplemento de la especificación USB 2.0 que amplía las capacidades de los dispositivos móviles y periféricos USB agregando la posibilidad de que estos funcionen como un Host USB (USB.org). La Figura 29 muestra el logotipo oficial del estándar USB On-The-Go (OTG).

Figura 29: Logotipo del estándar USB On-The-Go

Fuente: (USB.org, 2001)

b. Dispositivos de doble rol Un dispositivo con características OTG es, ante todo, un periférico USB compatible con la especificación USB 2.0. Los dispositivos de doble rol (*dual-role*, en inglés), además de ser compatibles con el USB 2.0, poseen las siguientes características (USB.org, 2001):

- Funcionalidad limitada de anfitrión USB (véase *"Anfitrión USB "*, pág. 25)
- Funcionamiento en *FS-USB* como dispositivo periférico (el funcionamiento en *HS-USB* es opcional)
- Soporte de *FS-USB* como anfitrión (el soporte de LS y HS es opcional)
- Soporta el *Session Request Protocol* (SRP)
- Soporta el *Host Negotiation Protocol* (HNP)
- Poseen un solo receptáculo tipo Mini-AB

IV. DELIMITACIÓN E IMPACTO DEL TEMA

El propósito principal de este estudio es fomentar el desarrollo tecnológico de nuestro país a través de la implementación de un proceso de investigación estructurado. Además, el estudio pretende continuar con una línea de investigación y desarrollo universitaria relacionada a la robótica asistencial. Los resultados obtenidos representan un aporte a la comunidad científica *delvalleriana* y a futuros proyectos relacionados con la robótica, el reconocimiento de entorno y la exploración de ambientes desconocidos.

La interfaz de datos implementada permitirá utilizar el radar láser Hokuyo URG-04LX-UG01como instrumento de medición en sistemas de reconocimiento de entorno en 2 dimensiones. Gracias a su largo alcance y exactitud, el radar láser podrá reemplazar a los sensores de distancia ultrasónicos o infrarrojos utilizados en la mayoría de aplicaciones robóticas para detectar o evadir obstáculos. Además, gracias a su alta resolución, es capaz de proveer suficiente información de su entorno para crear modelos bi-dimensionales de alta resolución del mismo.

Por otro lado, gracias a su alta frecuencia de muestreo, el radar láseres capaz de proveer grandes cantidades de información acerca del entorno varias veces por segundo y resulta útil en aplicaciones donde el tiempo de respuesta del vehículo robótico es vital. Por ejemplo, en vehículos aéreos o vehículos terrestres que se desplacen rápidamente.

Como parte de la delimitación del tema es importante mencionar que, en ningún momento, se modificaron o alteraron los componentes de hardware o software del sensor láser URG-04LX-UG01. En cambio, se utilizó la interfaz de datos USB 2.0 del sensor láser para comunicarse con un anfitrión (*Host*) USB dedicado implementado en el microcontrolador PIC24FJ64GB002. Para implementar dicho anfitrión USB se utilizó el módulo USB On-The-Go del microcontrolador.

La interfaz de datos permite controlar el radar láser desde un dispositivo externo a través del protocolo RS-232 utilizando el módulo UART integrado en el microcontrolador PIC24FJ64GB002.

Por otro lado, durante el desarrollo de la interfaz de datos del radar láser se utilizó como base el ejemplo "*USB Host - CDC - Serial Demo*" que forma parte de la librería "*USB Framework"*v2.9d de Microchip® para implementar el anfitrión USB en el microcontrolador. Dicha librería forma parte del paquete de librerías "*Microchip ApplicationLibraries"*v2012-02-15 y provee subrutinas y definiciones que permiten enumerar, controlar y monitorear el estado de los dispositivos del tipo *Communications Device Class* conectados al bus USB.

Además, se utilizó la especificación del protocolo de comunicación del radar láser, *SCIP* v2.0, como referencia para implementar los algoritmos de decodificación y detección de errores requeridos para procesar los datos de distancia en el microcontrolador.

A continuación se explicarán los criterios de selección del sensor láser utilizado en la implementación del sistema de reconocimiento de entorno (Hokuyo URG-04LX-UG01). Además, se presenta como resultado una breve investigación acerca de su funcionamiento, caracterización y protocolo de comunicación.

Luego, se explica la necesidad de implementar un anfitrión USB dedicado en un microcontrolador y se presenta el procedimiento llevado a cabo, los recursos de software utilizados, la estructura del algoritmo desarrollado, la circuitería eléctrica empleada y los resultados obtenidos.

Posteriormente, se explica la implementación de los algoritmos de recepción, decodificación y clasificación de los datos de distancia. Se explican detalladamente las técnicas empleadas para la detección y manejo de errores en la transmisión de los datos.

Por último, al final de la tesis se explica el algoritmo de recepción de los datos de distancia y se presenta la interfaz gráfica desarrollada. Esta permite visualizarlos datos de distancia del sensor láser en forma de radar y configurar sus parámetros de funcionamiento gráficamente.

V. DISEÑO EXPERIMENTAL

El diseño experimental de este trabajo de graduación está compuesto por las etapas mostradas en el diagrama de bloques de la Figura 30.

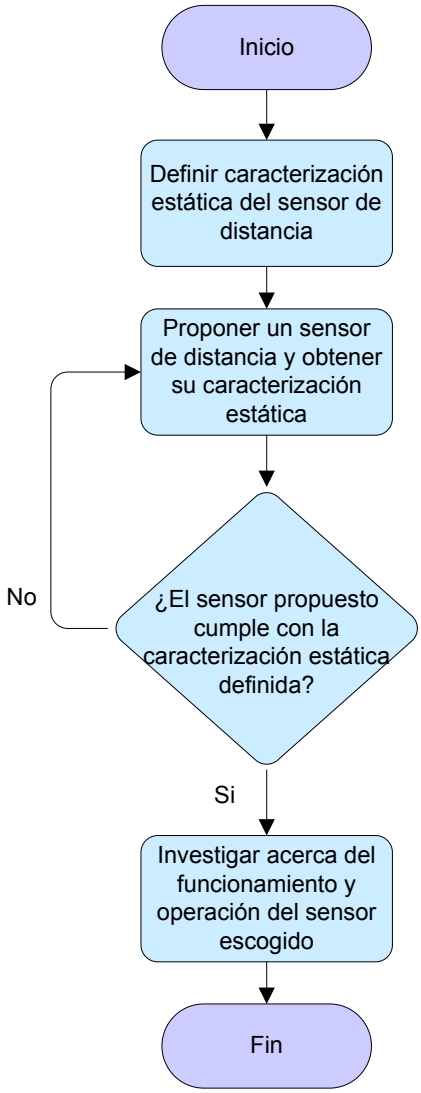
Figura 30: Diseño experimental de la investigación

Debido a la diversidad en la metodología empleada durante las etapas de esta tesis se propuso un diseño experimental que describe la serie de pasos secuenciales llevados a cabo para cumplir con los objetivos de la investigación. En los capítulos posteriores se explica detalladamente el procedimiento llevado a cabo en cada etapa del diseño experimental propuesto y se discuten los resultados obtenidos.

VI. ELECCIÓN DE UN SENSOR DE DISTANCIA

En este capítulo se describe el proceso de selección del sensor de distancia utilizado en la implementación del sistema de reconocimiento de entorno para aplicaciones robóticas. El diagrama de flujo de la Figura 31 muestra el diseño de este experimento.

Figura 31: Diseño experimental etapa 1



El sensor de distancia requerido para implementar el sistema de reconocimiento de entorno debía tener un alcance de varios metros de distancia y una resolución de pocos centímetros. Además, debía poder refrescar los datos de distancia varias veces por segundo y efectuar mediciones por lo menos 180 ° a su alrededor. El apartado siguiente muestra le caracterización estática requerida del sensor de distancia.

A. Caracterización del sensor de distancia

La Tabla 15 muestra la caracterización estática requerida del sensor de distancia; se muestra el valor requerido para cada característica.

Tabla 15: Caracterización estática del sensor de distancia

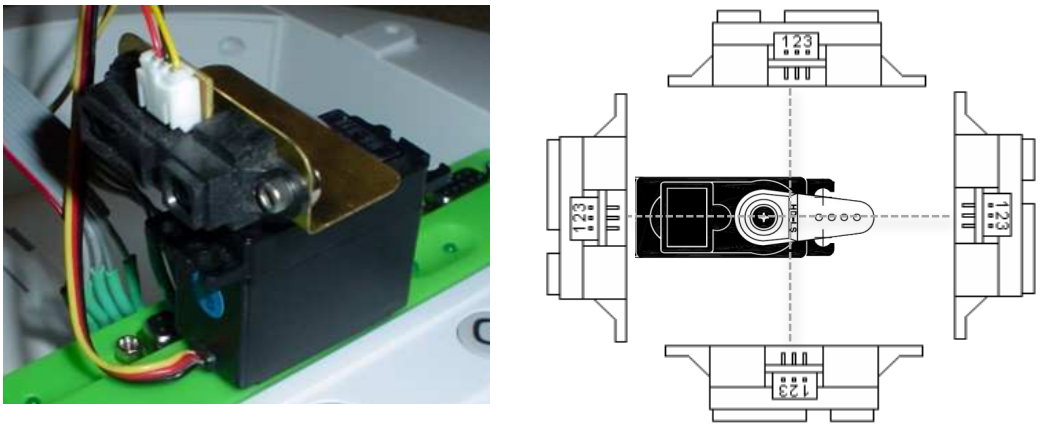
Característica	Descripción
Tipo de salida	Analógica, digital
Tipo del instrumento	Transmisión
Exactitud	±5%
Rango mínimo	10 cm
Rango máximo	1 m
Umbral	10 cm
Espacio muerto	0 – 10 cm
Resolución lineal	1 cm
Resolución angular	1°
Rango de detección	180°
Tiempo de barrido	500 ms
Frecuencia	2 Hz

B. Comparación de sensores de distancia

En la siguiente fase del experimento se propusieron 2 sensores de distancia de distinto tipo y se obtuvieron sus características estáticas. Los sensores propuestos fueron el sensor infrarrojo Sharp GP2D12 y el sensor láser Hokuyo URG-04LX-UG01.

1. Sensor infrarrojo de distancia Sharp GP2D12 Las características estáticas del Sharp GP2D12 fueron obtenidas de su hoja de datos a excepción del tiempo de barrido y la frecuencia de muestreo que fueron obtenidos experimentalmente. Debido al diseño del sensor infrarrojo, fue necesario acoplarlo a un servo motor para poder realizar un "barrido" y efectuar mediciones de distancia a lo largo del rango de detección (véase Figura 32).

Figura 32: Montaje de los sensores infrarrojos Sharp GP2D12



Fuente: (Parallax, 2005)

Para reducir el tiempo de barrido y aumentar la frecuencia de muestreo se utilizaron 4 sensores Sharp GP2D12; todos acoplados al servo motor (véase Figura 32). El eje del servo motor giraba 90° en sentido anti-horario para barrer los 360° del rango de detección y las 4 salidas analógicas se procesaron simultáneamente para obtener los datos de distancia.

La Tabla 16 muestra la caracterización estática obtenida del sensor de distancia infrarrojo Sharp GP2D12.

Tabla 16: Caracterización estática del sensor de distancia Sharp GP2D12

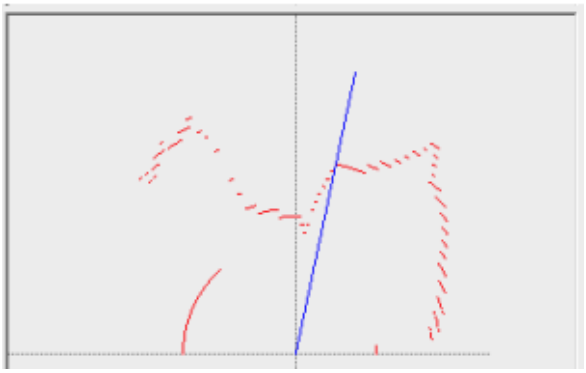
Característica	Descripción	Cumple con la caracterización
Tipo de salida	Analógica	✓
Tipo del instrumento	Transmisión	✓
Exactitud	±10%	✗
Rango mínimo	10 cm	✓
Rango máximo	80 cm	✗
Umbral	10 cm	✓
Espacio muerto	0 – 10 cm, >80 cm	✓
Resolución lineal	1 cm	✓
Resolución angular	1°	✓
Rango de detección	360°	✓
Tiempo de barrido	3.6 seg	✗
Frecuencia	0.28 Hz	✗

Fuente: (Society of Robots, 2005)

Según la Tabla 16, el sensor infrarrojo GP2D12 no cumple con todas las características estáticas definidas; su exactitud, rango máximo y sobre todo su frecuencia de muestreo no serían adecuados para la implementación de un sistema de reconocimiento de entorno en aplicaciones robóticas. Debido a la baja frecuencia de muestreo del sensor infrarrojo, el sistema de reconocimiento tardaría 3.6 segundos en realizar un barrido completo de medición y el tiempo de respuesta del sistema de control del robot sería demasiado alto para la mayoría de aplicaciones.

Además, debido a la poca exactitud del sensor infrarrojo, las representaciones en 2 dimensiones obtenidas no son lo suficientemente exactas (véase Figura 33).

Figura 33: Mapa en 2 dimensiones obtenido con el sensor infrarrojo



Entonces, según el diseño de la Figura 31, se descartó el sensor infrarrojo Sharp GP2D12 debido a que este no cumple con la caracterización estática definida.

2. Sensor láser de distancia Hokuyo URG-04LX-UG01 Según el diseño experimental de la etapa 1, se propuso un nuevo sensor: el radar láser Hokuyo URG-04LX-UG01 (véase Figura 34). Sus características estáticas fueron obtenidas de su hoja de datos. El sensor posee un haz de luz láser que gira en sentido anti-horario utilizando un motor eléctrico integrado y efectúa mediciones de distancia durante su recorrido.

Figura 34: Sensor láser Hokuyo URG-04LX-UG01

Fuente: (Hokuyo, 2009)

La Tabla 17 muestra la caracterización estática obtenida del sensor de distancia láser URG-04LX-UG01.

Tabla 17: Caracterización estática del sensor de distancia Hokuyo URG-04LX-UG01

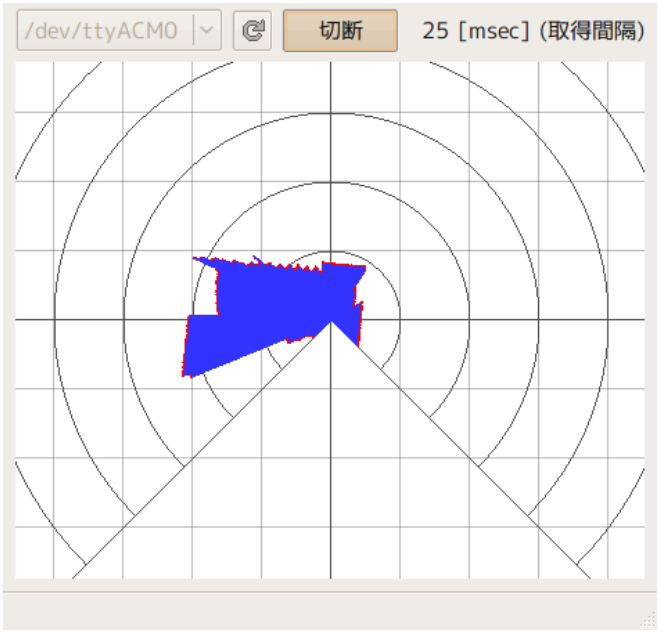
Característica	Descripción	Cumple con la caracterización
Tipo de salida	Digital	✓
Tipo del instrumento	Transmisión	✓
Exactitud	±3%	✓
Rango mínimo	2 cm	✓
Rango máximo	5.096 m	✓
Umbral	2 cm	✓
Espacio muerto	0 – 2 cm, >5.096 m	✓
Resolución lineal	1 mm	✓
Resolución angular	0.35°	✓
Rango de detección	240°	✓
Tiempo de barrido	100 ms	✓
Frecuencia	10 Hz	✓

Fuente: (Hokuyo, 2009)

Según la Tabla 17, el sensor láser URG-04LX-UG01 cumple con todas las características estáticas definidas; su exactitud, rango máximo y sobre todo su frecuencia de muestreo son adecuados para la implementación de un sistema de reconocimiento de entorno para aplicaciones robóticas. Debido a la alta frecuencia de muestreo del sensor láser, el sistema de reconocimiento refrescaría los datos de distancia del rango de detección hasta 10 veces por segundo (T = 100 ms) y el tiempo de respuesta del sistema de control del robot sería adecuado para la mayoría de aplicaciones.

Además, debido al largo alcance, exactitud y alta resolución del sensor láser, las representaciones en 2 dimensiones obtenidas con las mediciones de distancia son de alta resolución, detectan objetos más lejanos y son bastante exactas (véase Figura 35).

Figura 35: Mapa en 2 dimensiones obtenido con el sensor láser



Fuente: (Hokuyo, 2009)

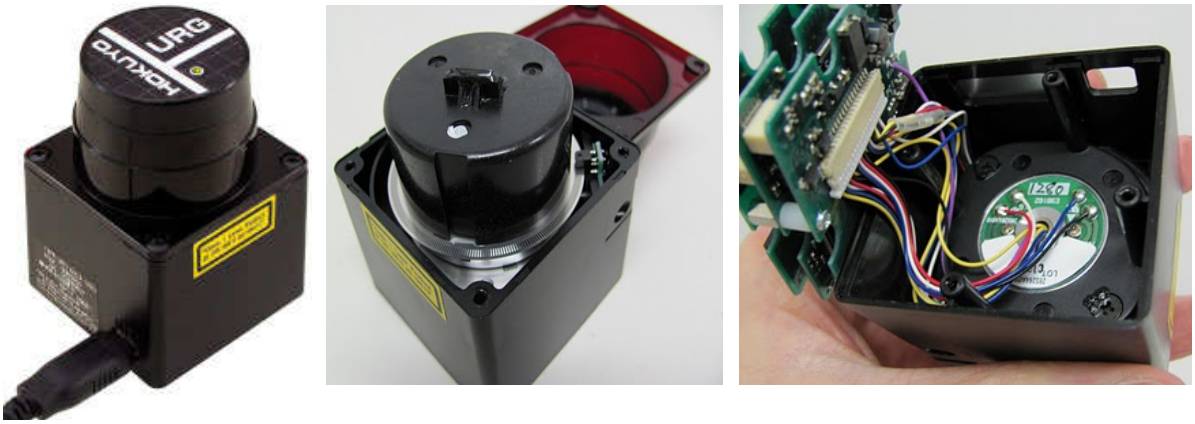
Entonces, según el diseño del experimento de la Figura 31, se escogió el sensor láser Hokuyo URG-04LX-UG01para la implementación del sistema de reconocimiento de entorno debido a que este cumple con la caracterización estática definida. Este sensor es un telémetro láser de barrido utilizado comúnmente en robots autónomos industriales y asistenciales (Hokuyo, 2009).

En el apartado siguiente se presenta, como resultado del experimento, una breve investigación acerca del funcionamiento y operación del sensor láser de distancia Hokuyo URG-04LX-UG01.

C. Hokuyo URG-04LX-UG01

1. Descripción general El URG-04LX-UG01 (véase Figura 36) es un telémetro o radar láser de largo alcance y alta resolución diseñado para interiores. Su fuente de luz es un láser infrarrojo con una longitud de onda de 785 nm que está clasificado como Clase 1 (véase *"Clase **1**"*, pág. 10). Su rango de detección es de 240° y puede medir distancias de hasta 5.6 metros. Además, es bastante ligero (160 g aprox.) debido a su *case* fabricado en policarbonato (Hokuyo, 2009).

Figura 36: Hokuyo URG-04LX-UG01

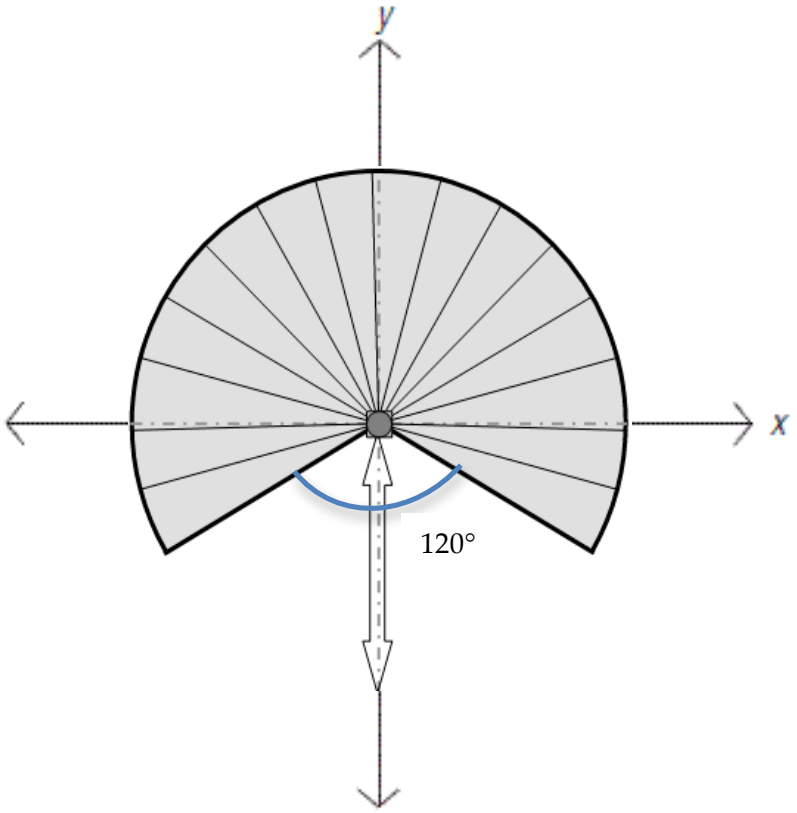


Fuente: (Hokuyo, 2009)

Su área de detección puede representarse como un semicírculo de 240° con un radio de 5 m. La parte trasera del sensor coincide con la parte negativa del eje vertical y la parte frontal con la parte positiva del mismo eje (véase Figura 37). Además, el "punto ciego" del radar es de 120° y está ubicado en su parte trasera (Hokuyo, 2009).

Por último, las mediciones de distancia son efectuadas utilizando el principio de medición basado en el desplazamiento de fase o "phase shift" (véase "Teoría de operación", pág. 5)

Figura 37: Área de detección del radar láser



Fuente: (Hokuyo, 2009)

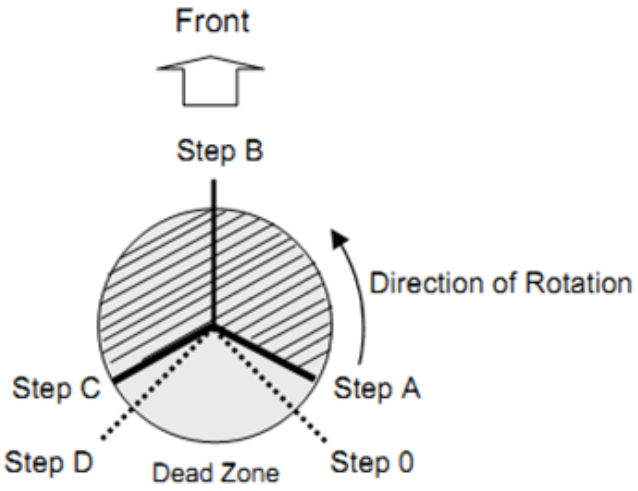
El radar URG-04LX-UG01 posee una resolución de 1 mm y una resolución angular de 0.35 °. La alta resolución y exactitud de este tipo de sensores láser provienen del diámetro angosto de su fuente de luz. El diámetro del rayo láser que utiliza el URG-04LX-UG01 es de 20 mm aproximadamente a una distancia de 2 metros y aumenta a 40 mm a una distancia de 4 metros. En general, el diámetro de los rayos láser es mucho más pequeño que el de otras fuentes de luz utilizadas en otro tipo de sensores fotoeléctricos de distancia (Hokuyo, 2009).

Por ejemplo, el sensor infrarrojo de distancia Sharp GPD2 utiliza un rayo infrarrojo como fuente de luz. El rayo tiene la forma de un ovoide y su parte más ancha, ubicada en la mitad del ovoide, mide 16 cm. Esto es, aproximadamente, 4 veces el diámetro del rayo láser del URG-04LX-UG01 (Society of Robots, 2005).

2. Funcionamiento y orientación de medición El radar láser Hokuyo URG-04LX-UG01 rota en dirección anti-horaria (visto desde arriba) y efectúa una medición de distancia en cada uno de los puntos de medición; estos reciben el nombre de "pasos" (Hokuyo, 2006). El radar tarda 100 ms en efectuar todas las mediciones a lo largo del rango de detección (Hokuyo, 2009).

En la Figura 38, el área sombreada representa el rango de detección del radar láser. Además, el paso '0' representa el primer punto en donde el radar puede efectuar una medición debido a limitantes mecánicas. Sin embargo, este no forma parte del rango de detección. El paso 'A' representa el primer punto de medición dentro del rango de detección; el paso 'B' está ubicado en el centro del rango de detección y coincide con el frente del sensor. El paso 'C' representa el último punto de medición del rango de detección; el paso 'D' es el último punto de medición y tampoco forma parte del rango de detección (Hokuyo, 2006).

Figura 38: Orientación de las mediciones



Fuente: (Hokuyo, 2006)

La Tabla 18 resume los parámetros de medición del URG-04LX-UG01 (Hokuyo, 2006).

Tabla 18: Parámetros de medición

Punto	Descripción	No. de paso	Dirección
Paso 0	Primer punto de medición	0	-45°
Paso A	Primer paso del rango de detección	44	-30°
Paso B	Paso frontal del sensor	384	90°
Paso C	Último paso del rango de detección	725	210°
Paso D	Último punto de medición	768	225°

Fuente: (Hokuyo, 2006)

La resolución angular del URG-04LX-UG01 está relacionada con la cantidad de sectores circulares en los que este dividida la circunferencia de medición según la Ecuación 2. Mientras más sectores circulares contenga la circunferencia, la resolución angular del radar aumentará (Hokuyo, 2006).

Ecuación 2: Resolución angular del radar URG-04LX-UG01

$$Resolución\ angular\ (°) = \frac{360°}{No.\ sectores\ circulares}$$

En el URG-04LX-UG01 la circunferencia de medición está dividida en 1024 sectores circulares. Entonces, podemos calcular la máxima resolución angular posible utilizando la Ecuación 2 (Hokuyo, 2006).

$$Resolución\ angular\ (°) = \frac{360°}{1024} = 0.3515° \cong 0.35°$$

Luego, podemos calcular la cantidad de puntos de medición o "pasos" que tendrá el radar láser utilizando la Ecuación 1.

$$No.de\ muestras = \frac{ángulo\ de\ barrido\ (°)}{resolución\ angular\ (°)} = \frac{240°}{0.3515°} \cong 682\ pasos$$

3. Caracterización estática del radar láser La Tabla 19 resume las principales características estáticas del radar láser Hokuyo URG-04LX-UG01 (Hokuyo, 2009). Para obtener mayor información véase la hoja de especificaciones del radar láser[1].

Tabla 19: Caracterización estática del URG-04LX-UG01

Característica	Descripción
Tipo de salida	Digital
Tipo del instrumento	Transmisión
Exactitud	± 30 mm a 1000 mm, ± 3% a 4000 mm
Rango	20 mm – 5096 mm
Umbral	20 mm
Espacio muerto	0 mm – 20 mm, > 5096 mm
Resolución	Lineal – 1 mm Angular – 0.35 mm (máxima)
Frecuencia	10 Hz (tiempo de barrido = 100 ms)

Fuente: (Hokuyo, 2009)

4. Interfaz de datos El URG-04LX-UG01 posee una interfaz de datos USB 2.0 para el procesamiento de los datos (véase Tabla 20).

Tabla 20: Interfaz de datos USB 2.0 del sensor láser

Versión	USB 2.0
Velocidad de transmisión	Full Speed - 12 Mbits/s (1.5 MB/s)
Clase de dispositivo USB	*Communications Device Class* (CDC)
VID / PID	0x15D1 / 0x0000
Soporte de controlador USB	Windows[2], Mac OS X, Linux

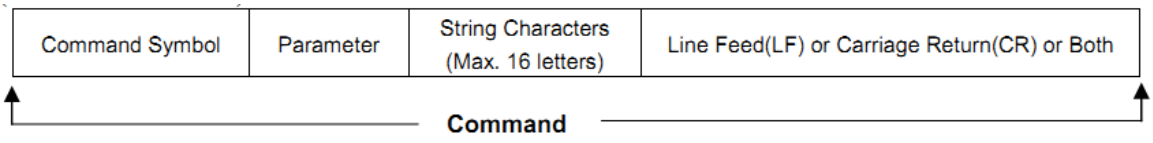
Fuente: (Hokuyo, 2009)

[1] Scanning Laser Range Finder URG-04LX-UG01(Simple URG) Specifications (Hokuyo, 2009)
[2] El controlador USB está disponible en: http://www.hokuyo-aut.jp/login/index.html

5. Protocolo de comunicación (SCIP2.0) El protocolo de comunicación utilizado por el URG-04LX-UG01 es el SCIP2.0. Este estándar fue desarrollado por un grupo de investigadores del laboratorio de Robótica Inteligente en la Universidad de *Tsukuba*, en Japón. Su objetivo principal era desarrollar una interfaz flexible y eficiente que permitiera utilizar esta serie de sensores en aplicaciones robóticas (Hokuyo, 2006).

a. Formato de comunicación El sensor y el Host intercambian información utilizando un grupo de comandos predefinidos en el protocolo SCIP2.0; esta sección explica el formato que deben tener dichos comandos. La comunicación debe ser iniciada por el Host enviando alguno de los comandos definidos utilizando el formato mostrado en la Figura 39 (Hokuyo, 2006).

Figura 39: Formato de comunicación Host -> sensor



Fuente: (Hokuyo, 2006)

La Tabla 21 explica los parámetros de los comandos enviados hacia el sensor.

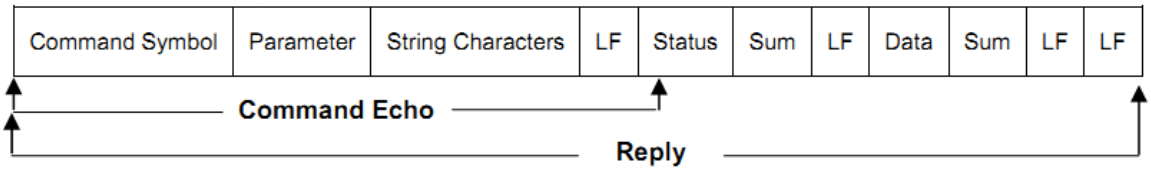
Tabla 21: Parámetros en un comando

Parámetro/Longitud	Descripción
Command Symbol (2 bytes)	Es un mnemónico que identifica a todos los comandos definidos en el protocolo SCIP2.0. Cada comando posee un mnemónico distinto.
Parameter	Es la información requerida por algunos comandos para modificar la configuración del sensor o solicitar datos; su longitud depende del comando empleado.
StringCharacters (hasta 16 bytes)	Es una cadena alfa-numérica de hasta 16 caracteres incluida opcionalmente en el comando para verificar que el sensor reciba el comando satisfactoriamente.
Line Feed (LF)/ Carr. Return (CR)	Es el código de terminación de un comando. Este código puede estar formado por LF (0xA) o CR (0xD) o ambos (0xA + 0xD).

Fuente: (Hokuyo, 2006)

La respuesta del sensor a un comando está formada por 2 partes: El eco o "repetición" del comando recibido y la respuesta a dicho comando (véase Figura 40). La Tabla 22 explica los parámetros de la respuesta del sensor a un comando.

Figura 40: Formato de comunicación sensor-> Host



Fuente: **(Hokuyo, 2006)**

Tabla 22: Parámetros de la respuesta del radar

Parámetro/Longitud	Descripción
STATUS (2 bytes)	Es un código de estado comprendido entre 0-99 que informa el estado del comando actual; indica errores si el comando recibido es indefinido, inválido o si está incompleto.
SUM (1 byte)	Es un carácter enviado por el sensor utilizado para autenticar los datos recibidos y detectar errores en la transmisión (véase *"Errores en la respuesta del sensor"*, pág. 60).
DATA (hasta 64 bytes)	Son los datos relacionados al comando actual; están agrupadosen segmentos dehasta 64 bytes; los segmentos están separados entre sí por la secuencia SUM +LF.
Line Feed (LF) (2 bytes)	Es el código de terminación de la respuesta del sensor; indica el final dela misma. Este código está formado por dos caracteres LF (0xA + 0xA).

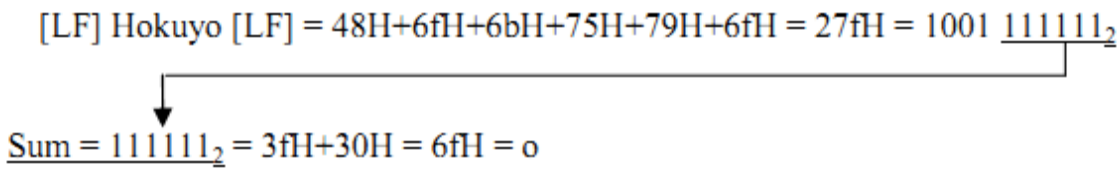
Fuente: (Hokuyo, 2006)

b. Detección de errores

1) Errores en la transmisión de un comando El campo de caracteres de cadena (*string characters*) descrito en la Tabla 21puede ser utilizado para verificar que el sensor reciba adecuadamente los comandos enviados desde el Host. Por ejemplo, cuando se requiere enviar el mismo comando varias veces, es posible incluir una cadena alfa-numérica distinta en cada comando y verificar su correcta recepción revisando el eco del comando en la respuesta del sensor. La cadena alfa-numérica debe estar separada por un punto y coma (;) al inicio para diferenciarla del campo "parámetros" (Hokuyo, 2006).

2) Errores en la respuesta del sensor Los errores en la transmisión de los datos contenidos en la respuesta del sensor pueden ser detectados utilizando el byte de SUM descrito en la Tabla 22. Este byte es calculado sumando los códigos ASCII de todos los caracteres comprendidos entre 2 caracteres LF (0xA), luego tomando los 6 bits menos significativos de la suma y sumando 0x30 a dicha cantidad (véase Figura 41).

Figura 41: Autenticación de los datos en la respuesta del sensor



Fuente: (Hokuyo, 2006)

3) Códigos de estado El URG-04LX-UG01 provee códigos de estado en el campo STATUS que pueden indicar errores en el funcionamiento del sensor o el formato de los comandos recibidos. La descripción de los códigos de estado más comunes se muestran a continuación. Los códigos distintos a 00 o 99 representan códigos de error (Hokuyo, 2006).

Tabla 23: Códigos de estado

Código de estado	Descripción
00	El comando fue recibido con éxito
21 ~ 49	El procesamiento se detuve para verificar un error
50 ~ 97	Problema con el hardware (*mal funcionamiento del láser o el motor*)
98	Resume el proceso luego de confirmar la operación normal del láser
0A	Incapaz de responder al comando
0B	Espacio insuficiente en el buffer de salida o se repitió un comando que ya se había procesado.
0C	El comando tiene parámetros insuficientes – 1
0D	Comando indefinido -1
0E	Comando indefinido -2
0F	El comando tiene parámetros insuficientes -2
0G	La cadena alfa-numérica en el comando excede los 16 caracteres
0H	La cadena alfa-numérica en el comando tiene caracteres inválidos

Fuente: **(Hokuyo, 2006)**

4) Errores de medición Las mediciones de distancia efectuadas por el sensor también pueden contener códigos de error (véase Tabla 24). Estos están codificados en los números menores al rango mínimo del radar (20 mm); es decir, entre 0 y 19 (Hokuyo, 2006).

Tabla 24: Códigos de error en los datos de medición

Código de error	Descripción
0	El objeto detectado se encuentra posiblemente a 22 metros
1~2	La luz reflectada tiene baja intensidad
6	Otros

Fuente: (Hokuyo, 2006)

Continuación Tabla 24

Código de error	Descripción
7	La medición efectuada en la paso anterior y posterior contienen errores
8	Diferencia de intensidad en 2 ondas
9	El mismo paso tuvo error en los últimos 2 barridos del sensor.
10~17	Otros
18	Error en la medición debido a un objeto excesivamente reflejante.
19	Es un paso no-medible.

Fuente: (Hokuyo, 2006)

c. Comandos del protocolo La interfaz de datos inalámbrica desarrollada para el URG-04LX-UG01 implementa todos los comandos definidos en el protocolo SCIP2.0. Estos están clasificados según su función en 3 grupos: Comandos de información del sensor, comandos de adquisición de distancias y comandos de configuración de funcionamiento.

A continuación se describirán brevemente todos los comandos definidos en el protocolo SCIP2.0. Sin embargo, se detallará el funcionamiento únicamente de los comandos empleados comúnmente en una aplicación robótica que utilice un radar láser como el URG-04LX-UG01. Para mayor información al respecto de todos los comandos consulte la especificación del protocolo SCIP2.0[3].

1) Comandos de información del sensor Este tipo de comandos son utilizados para obtener información acerca de la versión, estado actual y parámetros de funcionamiento del sensor láser. La Tabla 25 presenta la descripción de este tipo de comandos (Hokuyo, 2006).

[3] Communication Protocol Specification for SCIP2.0 Standard (Hokuyo, 2006)

Tabla 25: Descripción de los comandos de adquisición de información

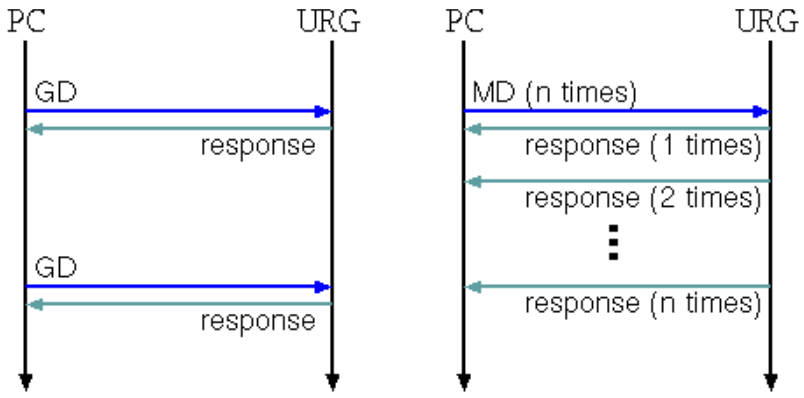
Mnemónico	Descripción	Formato
VV	El sensor transmite los detalles de su versión como número de serie, versión de *firmware*, versión del protocolo, entre otras.	"VV\n"
PP	El sensor transmite sus especificaciones como velocidad del motor, número de pasos, primer/último paso del rango de medición, entre otros.	"PP\n"
II	El sensor transmite su estado actual que incluye el estado del láser, la velocidad actual del motor, el modo de medición, diagnóstico de funcionamiento, entre otros.	"II\n"

Fuente: (Hokuyo, 2006)

2) Comandos de adquisición de distancias Este tipo de comandos son utilizados para obtener datos de medición del sensor láser. El protocolo SCIP2.0 define 4 comandos de adquisición de datos: GD, GS, MD, MS. Estos se diferencian según la forma en la que reciben los datos.

Los comandos GD/GS proporcionan los datos de un escaneo del radar, mientras que los comandos MD/MS proporcionan los datos de 'n' escaneos consecutivos según se especifique en los parámetros del comando (véase Figura 42).

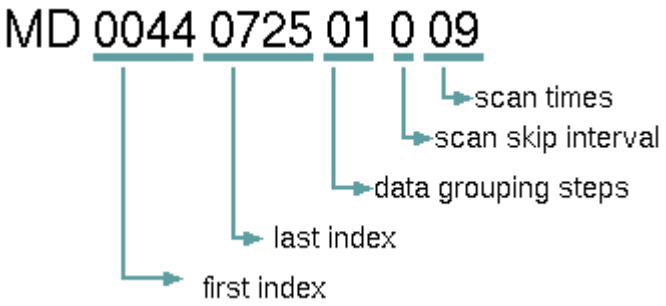
Figura 42: Comandos de adquisición de distancias, GD/GS (izq.) MD/MS (der.)



Fuente: (Hokuyo, 2012)

Parámetros de los comandos de adquisición de datos La Figura 43 muestra el formato general de los parámetros en los comandos de adquisición de datos MD/MS.

Figura 43: Parámetros de los comandos MD/MS



Fuente: (Hokuyo, 2012)

La Tabla 26 muestra la descripción de los parámetros de los comandos MD/MS.

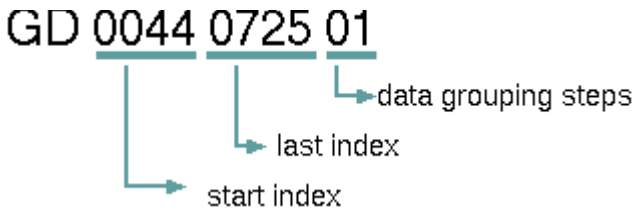
Tabla 26: Descripción de los parámetros de los comandos MD/MS

Parámetro/ Longitud	Descripción	Rango
COMMAND (2 bytes)	Representa el símbolo de comando; en este caso puede ser "MD" o "MS".	MS/MD
STARTING STEP (4 bytes)	Es el paso inicial de medición.	0044 – 0725
END STEP (4 bytes)	Es el paso final de medición; este debe ser mayor al paso inicial (STARTING STEP).	0044 –0725
CLUSTER COUNT (2 bytes)	Permite agrupar mediciones adyacentes en una sola para reducir la cantidad de datos transmitidos; modifica la resolución angular.	00 - 99
SCAN INTERVAL (1 byte)	Indica el número de barridos que se omiten de la transmisión continua de datos; reduce la frecuencia de transmisión de datos.	0 - 9
NUMBER OF SCANS (2 bytes)	Indica el número de barridos de datos requeridos. 00: el sensor enviará escaneos indefinidamente hasta recibir un comando QT.	00 - 99

Fuente: (Hokuyo, 2006)

Si el parámetro CLUSTER COUNT tiene un valor de 'n', las mediciones adyacentes son agrupadas en grupos de 'n' elementos y el resultado de la agrupación es la medición más pequeña del grupo (excluyendo mediciones con error). Por ejemplo, si el parámetro CLUSTER COUNT = 3 y las mediciones de 3 pasos adyacentes son: 3010 mm, 3050 mm y 3005 mm, el radar transmitiría únicamente la medición de 3005 mm. En este caso, la resolución angular sería entonces 0.35° * 3 = 1.05°. La Figura 44 muestra el formato general de los parámetros en los comandos de adquisición de datos GD/GS.

Figura 44: Formato de los parámetros en los comandos GD/GS



Fuente: (Hokuyo, 2012)

La Tabla 27 presenta la descripción de los 4 comandos de adquisición de datos definidos en el protocolo SCIP2.0.

Tabla 27: Comandos de adquisición de distancias

Mnemónico	Descripción	Rango
GS	El sensor transmite un barrido de medición; cada distancia está codificada en 2 caracteres ASCII.	20 – 4095 mm
GD	El sensor transmite un barrido de medición; cada distancia está codificada en 3 caracteres ASCII.	20 - 5600 mm
MS	El sensor transmite el número de barridos especificados en el parámetro NUMBER OF SCANS del comando; cada distancia está codificada en 2 caracteres ASCII.	20 – 4095 mm
MD	El sensor transmite el número de escaneos especificados en el parámetro NUMBER OF SCANS; cada distancia está codificada en 3 caracteres ASCII.	20 – 5600 mm

Fuente: (Hokuyo, 2006)

El número de bytes transmitidos por el sensor en el campo DATA de la respuesta a un comando de adquisición de datos puede calcularse con la Ecuación 3 (véase Figura 40).

Ecuación 3: Número de bytes transmitidos en la respuesta del sensor

$$No.\,de\ bytes\ transmitidos = \ No.\,de\ pasos * No.\,de\ ASCIIs\ de\ codificación$$

Por ejemplo, si la resolución angular del sensor se configura a 0.35° con CLUSTER COUNT = 1 y se utiliza el comando MS, que codifica cada distancia en 2 caracteres ASCII, entonces el número de bytes transmitidos por el sensor se calcula como:

$$No.\,de\ bytes\ transmitidos = \left(\frac{240°}{0.3515°}\right) * 2 = 1364\ bytes \cong 1.3\ kB$$

En la ecuación anterior, el número de pasos se calculó con la Ecuación 1. Además, la tasa de generación de datos puede calcularse según la Ecuación 4.

Ecuación 4: Tasa de generación de datos

$$Tasa\ de\ generación = \frac{No.\,de\ byte\ transmitidos}{tiempo\ de\ barrido} = \frac{1.3\ kB}{100\ ms} = 13\ kB/s$$

a) Comando MS Según la Tabla 27, el comando MS codifica las distancias en 2 caracteres ASCII en un rango comprendido entre 20 – 4095 mm. El sensor responde a este comando con el eco del comando recibido seguido del código de estado '00' y de la respuesta que contiene los datos de medición requeridos (Hokuyo, 2006).

El láser se enciende automáticamente antes de efectuar la medición y se apaga luego de completar el número de escaneos definidos en los parámetros del comando (Hokuyo, 2006). La Figura 45muestra el formato del comando MS; sus parámetros son los descritos en la Tabla 26.

Figura 45: Formato del comando MS

M (4dH)	D (44H) or S (53H)	Starting Step (4bytes)	End Step (4 bytes)	Cluster Count (2bytes)	Scan Interval (1 byte)
Number of Scans (2 bytes)	String Characters (max 16-letters)	LF (1 byte)			

Fuente: (Hokuyo, 2006)

Si el comando MS es recibido con error entonces la respuesta del sensor tendrá el formato mostrado en la Figura 46 y el campo STATUS contendrá el código de error correspondiente.

Figura 46: Respuesta del comando MS con error

M	D or S	Starting Step	End Step	Cluster Count	Scan Interval
Number of Scans	LF	String Characters	LF		
Status	Sum	LF	LF		

Fuente: (Hokuyo, 2006)

La Tabla 28 muestra la descripción de los códigos de error asociados a los comandos de adquisición de datos.

Tabla 28: Códigos de error en comandos de adquisición de datos

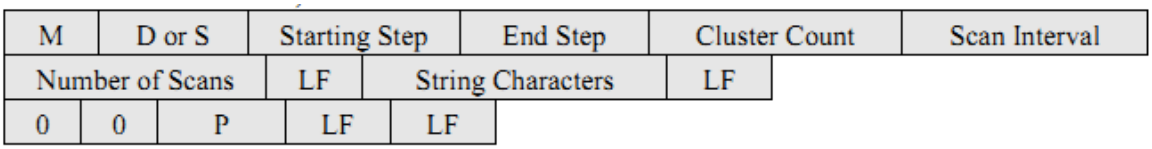
Código de error	Descripción del error
01	El paso inicial contiene un valor no-numérico
02	El paso final contiene un valor no-numérico
03	El parámetro CLUSTER COUNT tiene un valor no numérico
04	El paso final está fuera del rango de detección
05	El paso final es menor al paso inicial
06	El parámetro SCAN INTERVAL tiene un valor no numérico
07	El parámetro NUMBER OF SCANS tiene un valor no numérico

Fuente: (Hokuyo, 2006)

Por otro lado, si el comando MS es recibido sin errores entonces la respuesta del sensor iniciará con el encabezado mostrado en la Figura 47. Nótese que esta respuesta contiene el código de estado '00' indicando que el comando fue recibido con éxito.

Este encabezado será transmitido una única vez al inicio de la respuesta a un comando de adquisición de datos múltiples (i.e. MD, MS).

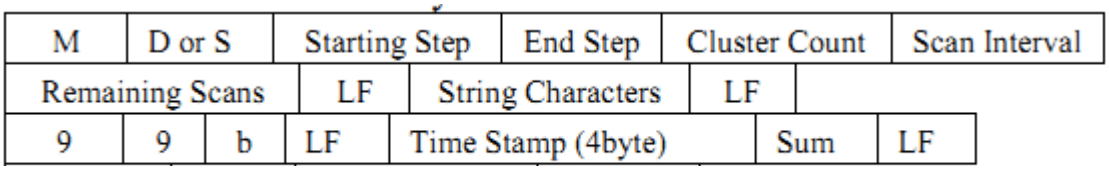
Figura 47: Respuesta del comando MS sin errores



Fuente: (Hokuyo, 2006)

Luego, el sensor transmitirá los datos del número de barridos de medición especificado en el parámetro NUMBER OF SCANS del comando. Al inicio de cada barridoo de medición el sensor transmitirá el encabezado mostrado en la Figura 48.

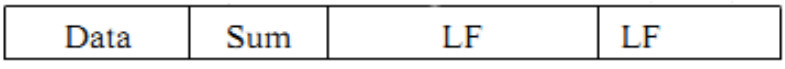
Figura 48: Encabezado de los datos de medición



Fuente: (Hokuyo, 2006)

El campo REMAINING SCANS (2 bytes) indica el número de barridos de medición pendientes de transmitir. Luego, el sensor transmitirá los datos de medición del barrido actual agrupados en segmentos de hasta 64 bytes separados entre sí por la secuencia SUM + LF. Si la cantidad de datos transmitidos es menor a 64 bytes, estos tendrán el formato mostrado en la Figura 49.

Figura 49: Formato de los datos de medición menores a 64 bytes



Fuente: (Hokuyo, 2006)

Si la cantidad de datos transmitidos es mayor a 64 bytes y termina sin bytes restantes, estos tendrán el formato mostrado en la Figura 50.

Figura 50: Formato de los datos de medición mayores a 64 bytes

Data Block 1 (64 byte)	Sum	LF	
----------	Sum	LF	
Data Block N (64 byte)	Sum	LF	LF

Fuente: (Hokuyo, 2006)

Si la cantidad de datos transmitidos es mayor a 64 bytes y termina con bytes restantes, estos tendrán el formato mostrado en la Figura 51.

Figura 51: Formato de los datos de medición mayores a 64 bytes con bytes restantes

Data Block 1 (64 byte)	Sum	LF	
----------	Sum	LF	
Data Block N-1 (64 byte)	Sum	LF	
Data Block N (n byte)	Sum	LF	LF

Fuente: (Hokuyo, 2006)

El proceso es repetido hasta que se hayan transmitido todos los datos. Nótese que en la transmisión del último barrido de medición el encabezado del último grupo de datos contendrá el parámetro REMAINING SCANS con un valor igual a 0 indicando que no hay más datos disponibles.

3) Comandos de configuración de funcionamiento Este tipo de comandos son utilizados para modificar las características de funcionamiento y operación de los radares láser como encender/apagar el láser, modificar el *baudrate*, entre otras (Hokuyo, 2006). La Tabla 29 presenta la descripción de los comandos de configuración definidos en el protocolo SCIP2.0.

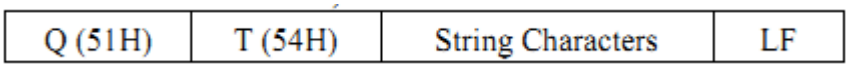
Tabla 29: Comandos de configuración

Mnemónico *(command symbol)*	Descripción	Formato del comando
SCIP2.0	Cambia la versión del protocolo de SCIP 1.x a SCIP 2.x	"SCIP2.0\n"
TM0, TM1, TM2	Cambia el modo de operación de la estampa de tiempo "*time stamp*".	"TMx\n"
SS	Modifica el *baudrate* de comunicación de la interfaz de datos RS-232 del sensor	"II\n"
BM	Enciende el láser del sensor y habilita el modo de medición de distancias	"BM\n"
QT	Apaga el láser del sensor y deshabilita el modo de medición de distancias	"QT\n"
RS	Restablece a sus valores originales todos los parámetros que fueron modificados desde que el sensor fue encendido por última vez	"RS\n"
CR	Modifica la velocidad del motor interno del sensor láser	"CR\n"

Fuente: (Hokuyo, 2006)

a) Comando QT Según la Tabla 29, el comando QT Apaga el láser del sensor y deshabilita el modo de medición de distancias. El sensor responde a este comando con el eco del comando recibido seguido del código de estado '00'. La Figura 52 muestra el formato del comando QT (Hokuyo, 2006).

Figura 52: Formato del comando QT



Fuente: (Hokuyo, 2006)

Luego, si el comando MS es recibido sin errores entonces la respuesta del sensor tendrá el formato mostrado en la Figura 53. Nótese que esta respuesta contiene el código de estado '00' indicando que el comando fue recibido con éxito.

Figura 53: Respuesta del comando QT

Q	T	String Characters	LF	0	0	P	LF	LF

Fuente: (Hokuyo, 2006)

El comando QT es utilizado usualmente para deshabilitar la transmisión indefinida de escaneos de medición cuando se utiliza el comando MS con el parámetro NUMBER OF SCANS = 00 (véase *Tabla 26: Descripción de los parámetros de los comandos MD/MS*).

VII. IMPLEMENTACIÓN DE UNA INTERFAZ DE DATOS PARA EL SENSOR URG-04LX-UG01

A. Motivación

Según el estándar USB, los dispositivos periféricos no pueden comunicarse entre sí. Estos pueden comunicarse únicamente con el anfitrión USB que controla el bus de comunicación (Otten K. , 2008). El URG-04LX-UG01 funciona como un dispositivo periférico USB del tipo *Communications Device Class* (CDC) (véase *Tabla 20: Interfaz de datos USB 2.0 del sensor láser*). Por esta razón, requiere ser controlado desde un anfitrión USB con soporte para dispositivos de tipo CDC; por ejemplo, desde una computadora.

Sin embargo, en la mayoría de aplicaciones robóticas no es factible incluir una computadora en el diseño del robot debido a limitantes de peso, espacio y costo. Por ello surge la necesidad de implementar un anfitrión USB (*USB Host*, en inglés) en otro tipo de microarquitectura, en un dispositivo más ligero, pequeño y costeable; como por ejemplo, en un microcontrolador.

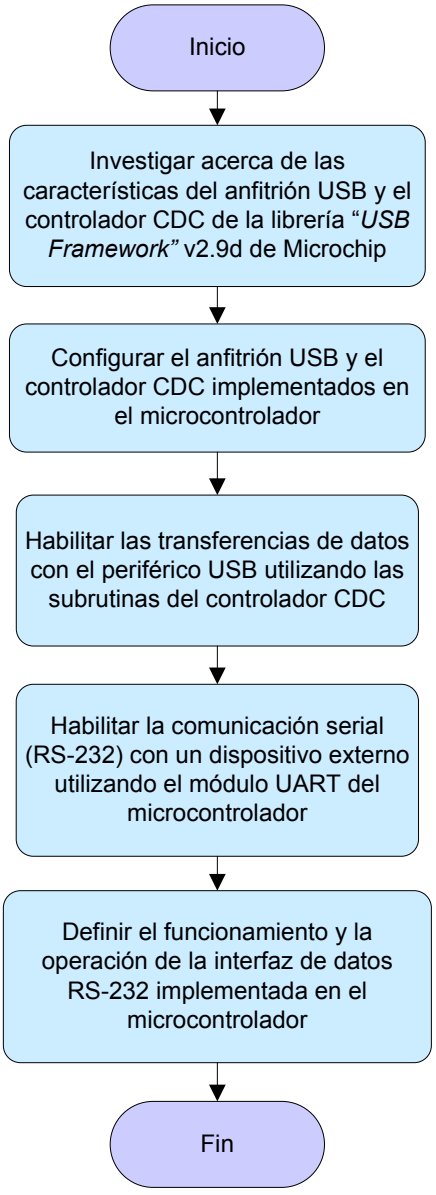
Afortunadamente, con la introducción de los microcontroladores de Microchip con el módulo USB *On-The-Go* (USB OTG), ahora es posible controlar la amplia gama de dispositivos periféricos USB existentes implementando un anfitrión USB en uno de estos microcontroladores. Entonces, el anfitrión USB implementado en el microcontrolador se encarga de ejecutar el controlador o *driver* USB adecuado para cada dispositivo periférico conectado al bus de comunicación (Otten K. , 2008).

Entonces, se planteó utilizar la interfaz de datos USB 2.0 del sensor láser URG-04LX-UG01 para comunicarse con un anfitrión USB dedicado implementado en un microcontrolador de 16 bits.

Luego, la interfaz serial de datos se implementó utilizando el módulo UART de dicho microcontrolador para habilitar la comunicación y las transferencias de datos entre el sensor láser y un dispositivo externo a través del protocolo RS-232.

En este capítulo se describe el proceso de implementación de una interfaz de datos para el sensor láser URG-04LX-UG01. El diagrama de flujo de la Figura 54muestra el diseño experimental de esta etapa.

Figura 54: Diseño experimental etapa 2



B. El anfitrión USB y el controlador o *driver* CDC

El anfitrión USB implementado es uno del tipo dedicado o limitado debido a los recursos limitados de memoria y procesamiento con los que cuenta un microcontrolador en comparación con los de una computadora. Las características de este tipo de anfitriones se describen en la sección: *"Anfitrión USB "*, pág. 25.

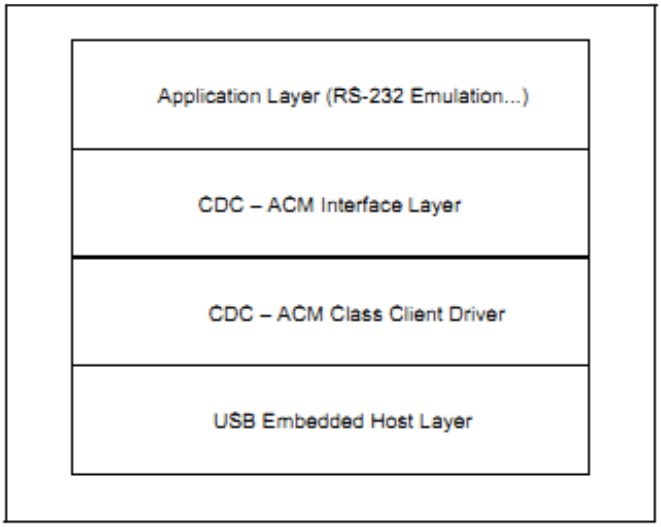
El anfitrión USB dedicado fue implementado utilizando la librería "USB Framework"(MCHPFSUSB) v2.9d de Microchip®. Esta librería forma parte del paquete de librerías "*Microchip Application Libraries"* v2012-02-15. Específicamente, se utilizó el ejemplo titulado: "USB Host - CDC - Serial Demo", contenido en la carpeta de instalación del paquete de librerías, como base para implementar el controlador CDC en el microcontrolador. Para implementar el anfitrión USB y el controlador CDC se utilizó el entorno de programación MPLAB IDE v8.84 y el compilador MPLAB C30 v3.31.

La librería *USB Framework* permite programar una aplicación que funciona como un controlador o *driver* USB para dispositivos del tipo *Communications Device Class* (CDC); este *driver* se ejecuta en el anfitrión USB dedicado implementado en el microcontrolador y permite comunicarse con el sensor láser URG-04LX-UG01 a través de su interfaz de datos USB 2.0.El controlador CDC que se ejecuta en el microcontrolador posee las siguientes características:

- Generación de eventos de conexión/desconexión al bus de comunicación.
- Generación de eventos de transmisión y recepción de datos.
- Soporta la transmisión de datos hacia el dispositivo periférico CDC.
- Soporta la recepción de datos desde el dispositivo periférico CDC.
- Soporta las transferencias de control y las masivas (*bulk transfers*).
- Generación de notificaciones de error al usuario.

1. Estructura del anfitrión USB y el controlador CDC La Figura 55 muestra la estructura del controlador o *driver* CDC que se ejecuta en el microcontrolador. La funcionalidad de anfitrión USB está implementada mediante un *stack* de múltiples capas en donde distintos componentes de la librería utilizada contribuyen a distintas capas del *stack.*

Figura 55: Estructura del *stack* USB



Fuente: (Otten K. , 2008)

Cada una de las capas del stack está formada por uno o más archivos de código fuente (.c) o archivos de encabezado (.h) que contienen subrutinas y/o definiciones que contribuyen a distintas funciones del anfitrión USB dedicado o el controlador CDC implementado en el microcontrolador (véase Tabla 30).

a. Capa de aplicación Esta capa utiliza subrutinas y definiciones contenidas en todas las capas inferiores del *stack* para habilitar el intercambio de información con el sensor láser a través de su interfaz de datos USB 2.0. Contiene las subrutinas utilizadas para transmitir los comandos del protocolo SCIP2.0 hacia el sensor láser y recibir los datos de distancia a través de la interfaz USB 2.0.

Esta capa también contiene las subrutinas utilizadas para habilitar la comunicación serial (RS-232) con un dispositivo externo utilizando el módulo UART del microcontrolador.

b. Capa de anfitrión USB dedicado Esta capa provee soporte básico de anfitrión USB; no es necesario conocer el funcionamiento y configuración de esta capa para poder hacer uso de la capa de aplicación.

c. Capas de interface y clase CDC-ACM Estas capas manejan todas las transferencias relacionadas a los dispositivos CDC como enumeración y configuración. Estas también proveen funciones para utilizar la capa de aplicación.

La Tabla 30 muestra una descripción de los archivos de código fuente (.c) o archivos de encabezado (.h) contenidos en cada una de las capas del *stack* que implementa el anfitrión USB dedicado.

Tabla 30: Capas del *stack* que implementa el anfitrión USB con soporte CDC

Capa	Nombre de archivo	Descripción
Anfitrión USB dedicado (USB Embedded Host Layer)	usb_host.c	Provee soporte de anfitrión USB dedicado para todos los dispositivos; no provee soporte para una clase de dispositivo USB en específico.
	usb_host.h	Contiene definiciones requeridas para el funcionamiento del anfitrión USB dedicado. Define la interfaz con el controlador del anfitrión USB.
	usb.h, usb_ch9.h, usb_common.h, usb_hal.h, usb_hal_pic24.h	Otros archivos de encabezado de soporte USB.

Fuente: (Otten K. , 2008)

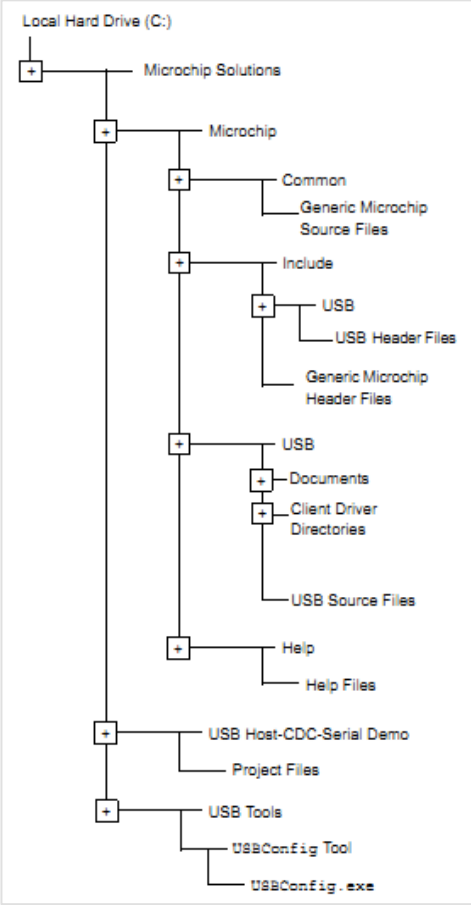
Continuación Tabla 30

Capa	Nombre de archivo	Descripción
Controlador/interfaz CDC (CDC Client/InterfaceLayer)	usb_host_cdc.c	Provee soporte para dispositivos periféricos de la clase CDC (*CommunicationsDeviceClass)* al anfitrión USB dedicado.
	usb_host_cdc.h	Contiene definiciones requeridas para el soporte de la clase CDC en el anfitrión USB dedicado. Define la interfaz con el controlador CDC.
	usb_host_cdc_interface.c	Provee funciones de interfaz que permiten acceder al controlador CDC desde la capa de aplicación.
	usb_host_cdc_interface.h	Contiene definiciones de interfaz para acceder al cliente CDC.
Aplicación (ApplicationLayer)	usb_config.c	Archivo de configuración del *stack* USB generado por la utilidad de configuración (*USBConfig.exe*).
	usb_config.h	Archivo de configuración del *stack* USB generado por la utilidad de configuración (*USBConfig.exe*).
	system.h	Contiene constantes del sistema referenciadas por otras librerías.
	HokuyoDrivervF.c	Contiene el código fuente de la aplicación principal.

Fuente: (Otten K. , 2008)

Todos los archivos de la Tabla 30 están contenidos en el directorio de instalación del paquete de librerías "*Microchip Application Libraries*": "C:\Microchip Solutionsvxxxx-xx-xx". La ruta de ubicación de los archivos de la librería utilizados para la implementación del anfitrión USB dedicado se muestra en la Figura 56.

Figura 56: Ruta de los archivos del *stack* USB



Fuente: (Gupta, 2009)

2. Arquitectura del anfitrión USB y el controlador CDC

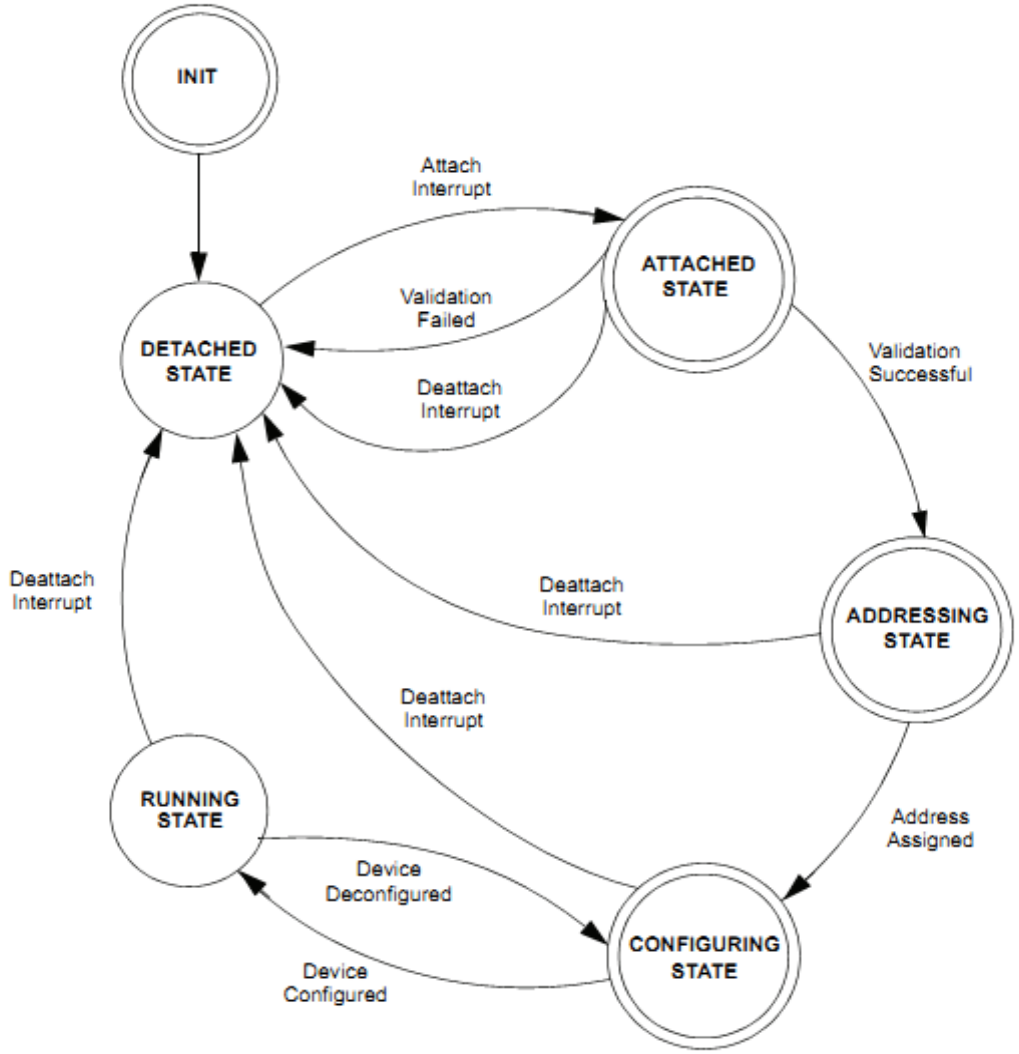
a. Inicialización del anfitrión USB dedicado La utilidad de configuración USB (USBConfig.exe) provee una macro de inicialización llamada "USBInitialize(x)" en el archivo generado "usb_config.h". Esta macro ejecuta todas las subrutinas de inicialización requeridas por la capa de anfitrión USB dedicado del *stack*.

Dentro de las subrutinas de inicialización que ejecuta la macro "USBInitialize(x)" están incluidas las de inicialización del módulo USB *On-The-Go* del microcontrolador utilizado (Gupta, 2009).

Entonces, la macro "USBInitialize(x)" debe ser llamada por lo menos una vez durante la etapa de inicialización de la aplicación principal (*HokuyoDrivervF.c*) para configurar e inicializar correctamente la capa del *stack* USB que implementa la funcionalidad de anfitrión USB dedicado.

b. Enumeración de los dispositivos periféricos CDC El *stack* de anfitrión USB dedicado implementa la máquina de estados finitos mostrada en la Figura 57para la enumeración de los dispositivos periféricos conectados al bus de comunicación. Esta genera eventos para indicar el estado de los dispositivos periféricos USB.

Figura 57: Máquina de estados finitos para el proceso de enumeración del *stack* USB



Fuente: (Otten K. , 2008)

Dichos eventos son atendidos en la capa de aplicación utilizando un manejador de eventos (*eventhandler*, en inglés) definido en el código fuente de la aplicación principal "*HokuyoDrivervF.c*" en la subrutina "*USB_ApplicationEventHandler*" (véase código fuente en el Apéndice).

La Tabla 31 muestra una descripción detallada de los eventos de interrupción generados en la capa de aplicación por la máquina de estados finitos de la Figura 57.

Tabla 31: Eventos generados por la máquina de estados finitos en la enumeración

Nombre del evento	Descripción
EVENT_VBUS_REQUEST_POWER	Indica que el dispositivo periférico conectado al bus está demandando demasiada potencia (>500 mA).
EVENT_VBUS_RELEASE_POWER	Indica el estado de la línea de alimentación VBUS del USB.
EVENT_HUB_ATTACH	Notificación de error que indica que se conectó un concentrador de puertos o *Hub* al bus de comunicación
EVENT_UNSUPPORTED_DEVICE	Notificación de error que indica que se conectó al bus un dispositivo periférico de una clase USB que no es soportada por el anfitrión USB dedicado.
EVENT_CANNOT_ENUMERATE	Notificación de error que indica que no fue posible enumerar al dispositivo periférico conectado al bus.
EVENT_CLIENT_INIT_ERROR	Notificación que indica un error en el proceso de inicialización del controlador CDC.
EVENT_OUT_OF_MEMORY	Notificación de error que indica que se agotó la memoria heap.
EVENT_UNSPECIFIED_ERROR	Notificación de error generada por distintos motivos.
EVENT_DETACH	Indica que el dispositivo periférico ha sido desconectado del bus de comunicación.

Fuente: (Otten K. , 2008)

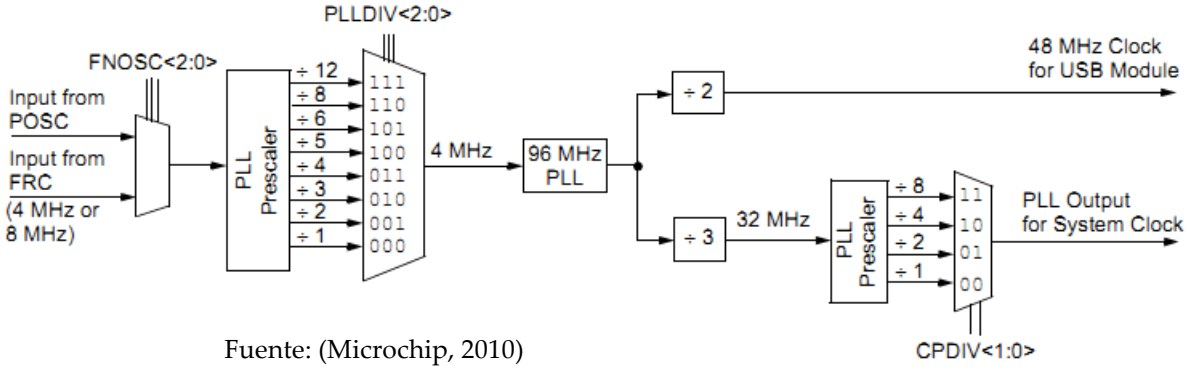
c. Operación normal de controlador CDC La utilidad de configuración USB (USBConfig.exe) también provee una macro llamada "USBTasks()" en el archivo generado "usb_config.h". Esta macro ejecuta todas las subrutinas que llevan a cabo tareas de procesamiento USB requeridas por la capa de anfitrión USB dedicado y el controlador CDC del *stack*.

Entonces, la macro "USBTasks()" debe ser ejecutada periódicamente en la aplicación principal (*HokuyoDrivervF.c*) para asegurar el funcionamiento adecuado del anfitrión USB y el controlador para dispositivos CDC.

3. Requisitos de procesamiento Para implementar el anfitrión USB dedicado en un microcontrolador fue necesario utilizar uno equipado con un módulo USB On-The-Go. El módulo USB OTG requiere un reloj de 48 MHz; esto es muy superior a la frecuencia máxima de osciladores comunes. Por esta razón, se utilizó un microcontrolador de la familia PIC24 que contara con un módulo multiplicador de frecuencia PLL incorporado.

Este multiplicador de frecuencia permite generar, a partir de un reloj de 4 MHz, un reloj de 96 MHz. Su salida es dividida en 2 ramas: una rama de 48 MHz para el módulo USB OTG y otra de 32 MHz utilizada para generar cualquiera de las siguientes frecuencias de reloj del sistema: 32 MHz, 16 MHz, 8 MHz, 4 MHz (véase Figura 58).

Figura 58: Configuración del multiplicador de frecuencia PLL

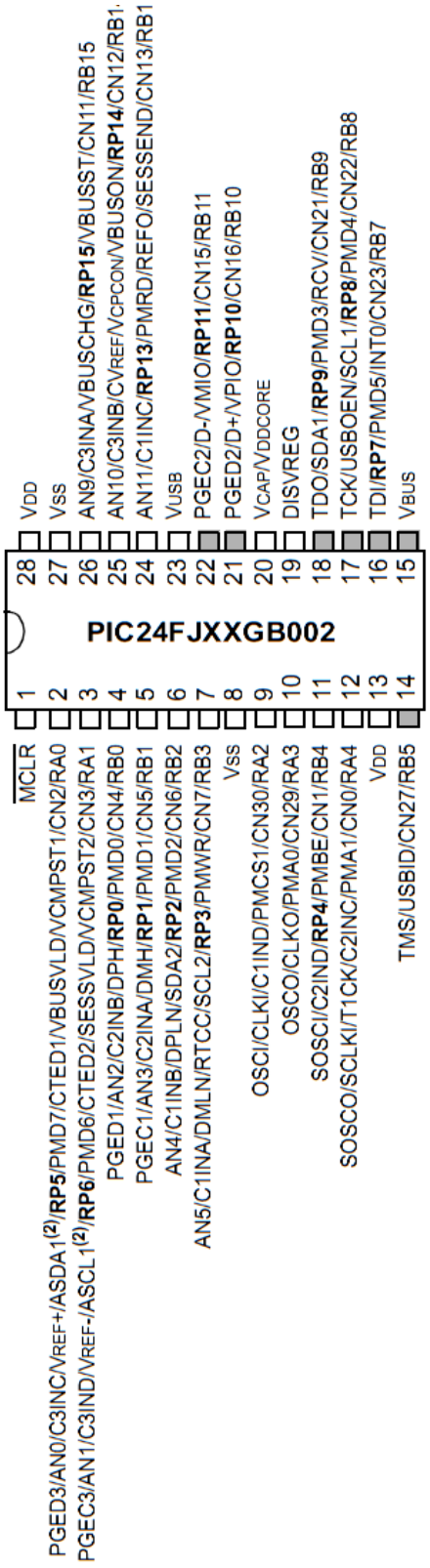


Fuente: (Microchip, 2010)

Además, era deseable que el microcontrolador tuviese un empaquetado DIP (Dual-In-Line)por su facilidad para soldarlo a una PCB. Por otro lado, era necesario que el microcontrolador contara con un módulo UART para poder habilitar la comunicación serial (RS-232) entre el sensor láser y un dispositivo externo.

Por lo anterior, se eligió el microcontrolador de 16 bits PIC24FJ64GB002 que cuenta con un módulo USB OTG, 1 módulo UART y un empaquetado SPDIP de 28 pines. La Figura 59 muestra el diagrama de patillaje del microcontrolador utilizado para implementar el anfitrión USB dedicado.

Figura 59: Patillaje del microcontrolador PIC24F64GB002



Fuente: (Microchip, 2010)

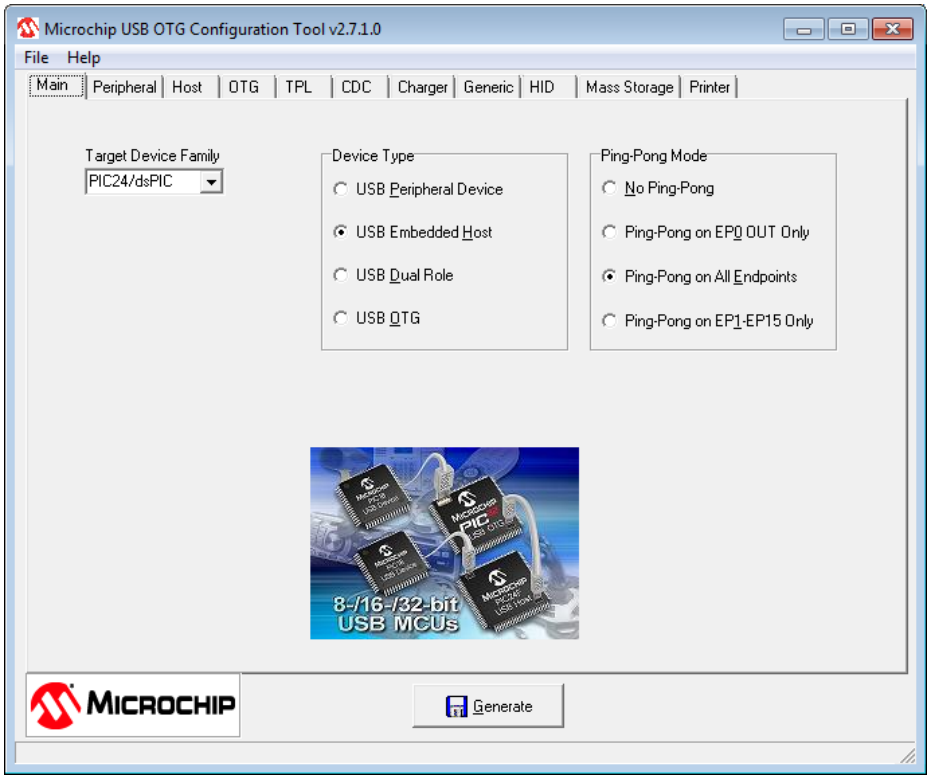
C. Implementación del anfitrión USB dedicado y el controlador o *driver* CDC

1. Configuración del controlador o *driver* CDC El controlador para dispositivos periféricos del tipo *Communications Device Class* (CDC) es instalado como parte del paquete de soporte para anfitriones USB que forma parte de la *Microchip Applications Libraries*.

Para configurar el controlador se utilizó la aplicación 'USBConfig.exe' que forma parte de la librería USB Framework y se localiza en la ruta de archivo:*"C:\Microchip Solutions v2010-10-19\USB Tools\USBConfigTool\USBConfig.exe"*.

En la pestaña "Main" de la aplicación 'USBConfig.exe' seleccione la familia de dispositivos "*PIC24/dsPIC*", en tipo de dispositivo seleccione "*USB Embedded Host*" y en modo de Ping-Pong seleccione "*Ping-Pong on All Endpoints*" (véase Figura 60).

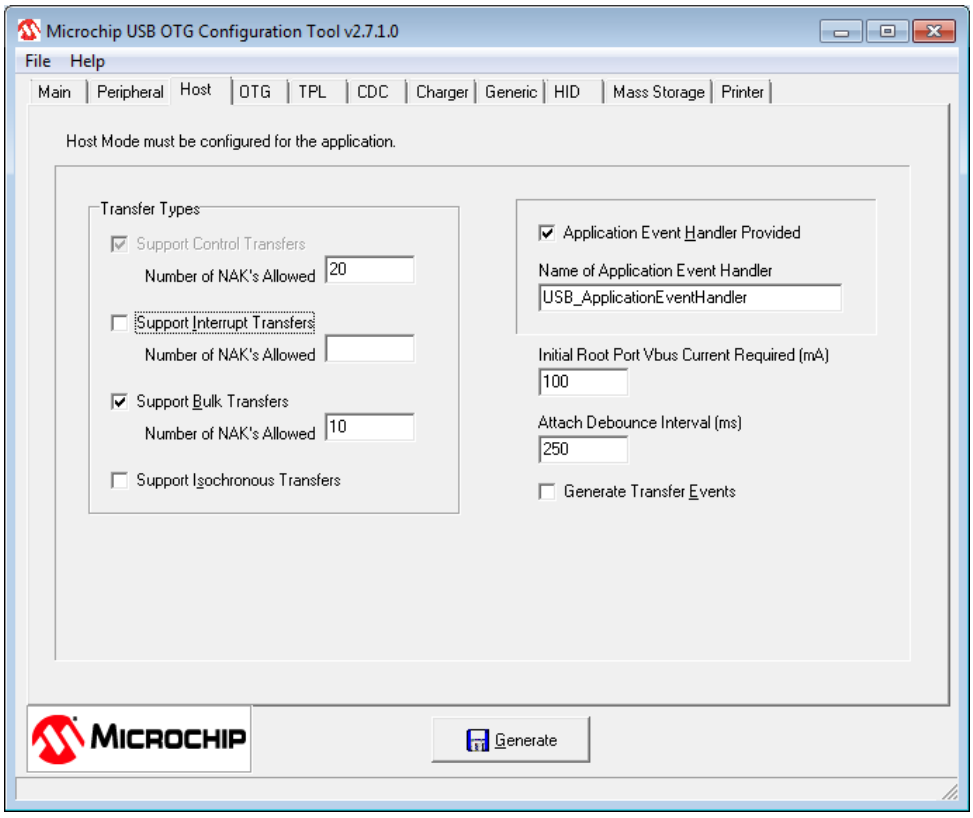
Figura 60: Configuración de la pestaña "Main"



Luego, en la pestaña "Host" se habilitaron únicamente las transferencias de control (*control transfers*) y las masivas (*bulk transfers*) debido a las características de los dispositivos periféricos de la *Communications Device Class* (CDC).

Además, se configuraron el número de paquetes NAK para las transferencias de control y las masivas, el manejador de eventos, la corriente inicial y tiempo de inserción como lo muestra la Figura 61.

Figura 61: Configuración de la pestaña "Host"



Luego, en la pestaña "CDC" se seleccionó la opción "*CDC Client is used in Host Mode*" para determinar el modo de funcionamiento del microcontrolador. Además, se configuraron los parámetros de la comunicación serial del anfitrión USB como lo muestra la Figura 62.

Figura 62: Configuración de la pestaña "CDC"

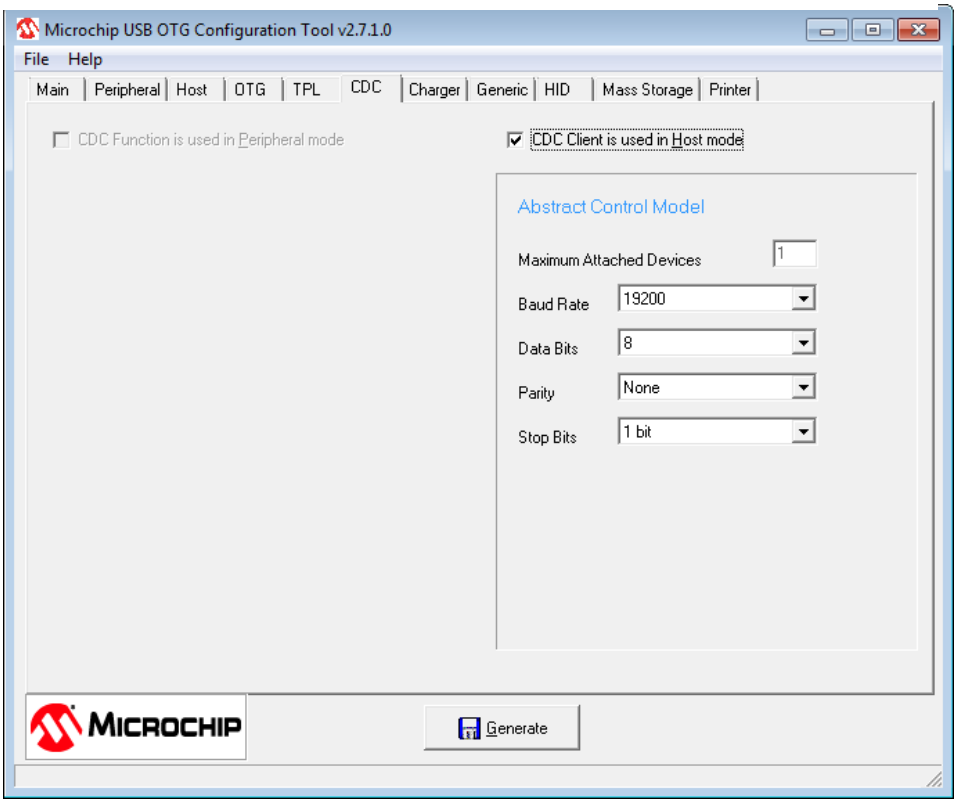
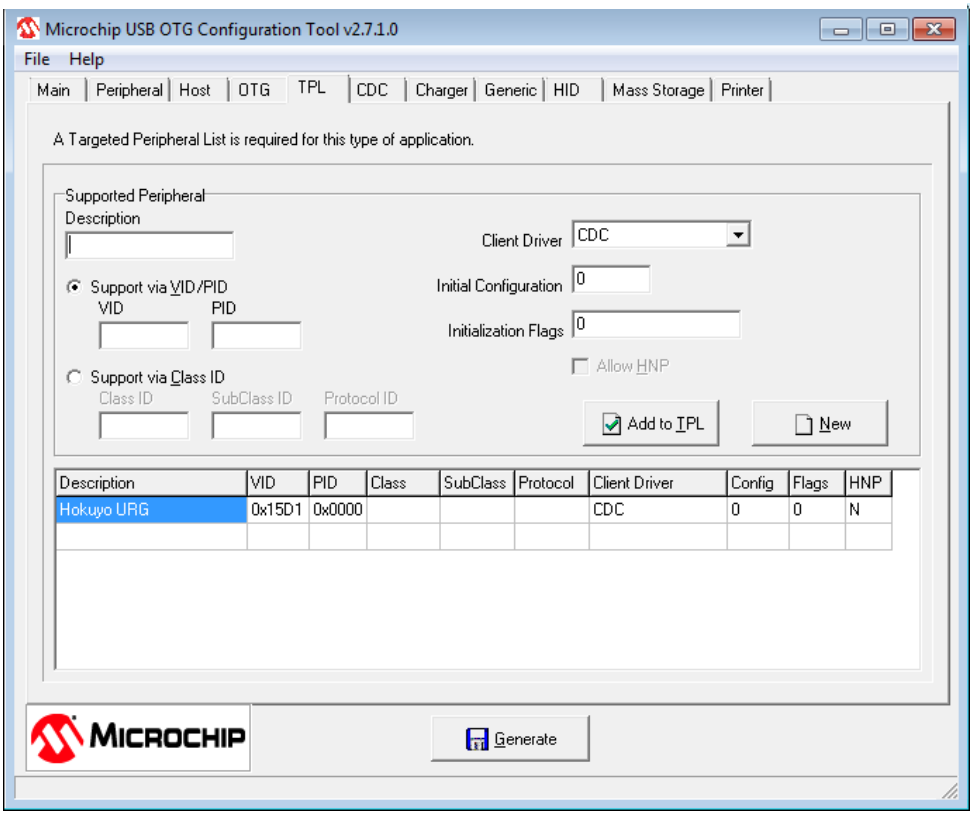


Figura 63: Configuración de la pestaña "TPL"



Por último, se agregó el radar láser como dispositivo periférico USB en la *Targeted Peripheral List* (TPL) en la pestaña "TPL" como lo muestra la Figura 63. Finalmente, es necesario presionar el botón "*Generate*" para generar los archivos de configuración "usb_config.c" y "usb_config.h". Ambos archivos deben ser copiados a la carpeta del proyecto de MPLAB IDE ya que forman parte de la capa de "aplicación" del *stack* que implementa el anfitrión USB en el microcontrolador (véase Tabla 30).

2. **Estructura del código fuente del anfitrión USB dedicado para dispositivos CDC** Como se mencionó anteriormente, la inicialización del anfitrión USB dedicado en la aplicación principal (*HokuyoDrivervF.c*) debe llevarse a cabo utilizando la macro "USBInitialize". Además, debe hacerse una llamada también a la macro "USBTasks" periódicamente para ejecutar todas las funciones básicas del anfitrión USB dedicado y el controlador CDC.

Ambas macros están definidas en el archivo de configuración "usb_config.h" generado por la utilidad de configuración (USBConfig.exe). Entonces, el código fuente de la aplicación principal (*HokuyoDrivervF.c*) debe tener la estructura mostrada en la Figura 64.

Figura 64: Estructura del código fuente en la aplicación principal

```
int main(void)
{
  [Initialization routines go here...]

  USBInitialize( 0 );

  while(1)
  {
    USBTasks();



    [main application code goes here..]
  }
}
```

Fuente: (Gupta, 2009)

3. **Transferencias de datos entre el anfitrión USB y el dispositivo periférico** Este apartado describe el proceso de implementación de las transferencias de datos entre el anfitrión USB y el sensor láser a través de su interfaz de datos USB 2.0.

Para ello se implementó una máquina de estados finitos en la capa de aplicación del *stack* USB; específicamente, en al archivo de la aplicación principal (*HokuyoDrivervF.c*) del microcontrolador. Dicha máquina de estados finitos controla la aplicación principal utilizando subrutinas y definiciones de la librería *USB Framework* (MCHPFSUSB) contenidas en capas inferiores del *stack* USB.

La máquina de estados finitos que controla la aplicación principal posee las siguientes características:

- Indica y actualiza el estado del dispositivo periférico CDC conectado al bus de comunicación.

- Transmite notificaciones del estado de las transferencias de datos pendientes o acerca de los errores en la transmisión de datos.

- Administra las transferencias de datos salientes (OUT transfer) desde el microcontrolador (anfitrión USB) hacia el sensor láser URG-04LX-UG01 (periférico CDC) a través del bus de comunicación.

- Administra las transferencias de datos entrantes (IN transfer) desde el sensor láser URG-04LX-UG01 (periférico CDC) hacia el microcontrolador (anfitrión USB) a través del bus de comunicación.

La Figura 65 muestra el diagrama de estados de la máquina de estados finitos implementada en la aplicación principal del microcontrolador.

Figura 65: Máquina de estados finitos de la aplicación principal

Fuente: (Gupta, 2009)

La máquina de estados finitos que controla la aplicación principal posee 8 estados que describen el estado actual de la aplicación (véase Figura 65). La Tabla 32 presenta una descripción detallada de cada uno de ellos.

Tabla 32: Estados de la máquina de estados finitos de la aplicación principal

Nombre de estado	Descripción
DEMO_INITIALIZE	Es el estado inicial de la máquina de estados finitos.
DEVICE_NOT_CONNECTED	Indica que el periférico USB no está conectado al bus.
DEVICE_CONNECTED	Indica que el periférico USB ha sido conectado al bus y enumerado satisfactoriamente.
READY_TO_TX_RX	Es el estado central de la máquina de estados finitos e indica que el periférico USB está listo para transmitir/recibir información.
SEND_OUT_DATA	Indica que se colocará una petición de transferencia de datos salientes (OUT request).
SEND_OUT_DATA_WAIT	Indica que la petición de transferencia de datos salientes fue colocada exitosamente y la aplicación está en espera de que se complete dicha transferencia.
GET_IN_DATA	Indica que se colocará una petición de transferencia de datos entrantes (IN request).
GET_IN_DATA_WAIT	Indica que la petición de transferencia de datos entrantes fue colocada exitosamente y la aplicación está en espera de que se complete dicha transferencia.

Fuente: (Gupta, 2009)

Además, la máquina de estados finitos utiliza algunas subrutinas definidas en la capa "*CDC-Class Client Driver*" del *stack* para diversas tareas como: programar una transferencia de datos de entrada/salida, determinar si la transferencia de datos actual ha concluido o detectar un nuevo dispositivo periférico USB conectado al bus de comunicación (véase Figura 65).

La Tabla 33 presenta una descripción detallada de las subrutinas de la capa "*CDC-Class Client Driver*" utilizadas en la máquina de estados finitos de la aplicación principal.

Tabla 33: Subrutinas de la capa "*CDC-Class Client Driver*" de la aplicación principal

Subrutina	Tipo de retorno	Descripción
USBHostCDC_ApiDeviceDetect	BOOL	Es utilizada para obtener el estado del periférico USB. Si el periférico ha sido enumerado satisfactoriamente la subrutina devuelve "TRUE".
USBHostCDC_Api_Get_IN_Data	BOOL	Es utilizada para realizar una petición de una transferencia de datos entrantes de hasta 64 bytes. Si la petición fue realizada satisfactoriamente la subrutina devuelve "TRUE".
USBHostCDC_Api_Send_OUT_Data	BOOL	Es utilizada para realizar una petición de una transferencia de datos salientes de hasta 64 bytes. Si la petición fue realizada satisfactoriamente la subrutina devuelve "TRUE".
USBHostCDC_ApiTransferIsComplete	BOOL	Es utilizada para obtener el estado de la transferencia de datos actual de entrada/salida. Mientras la transferencia de datos actual no se haya completado la subrutina devuelve "FALSE". Cuando se completa la transferencia, entonces devuelve "TRUE".

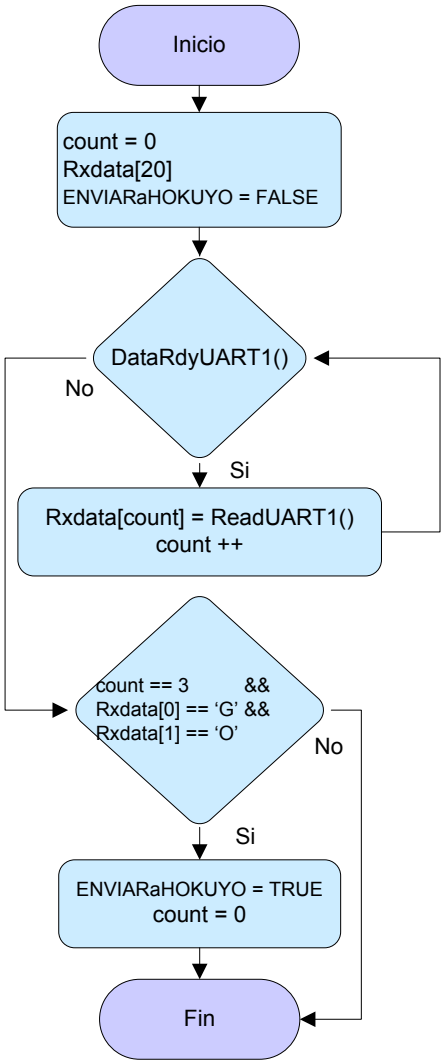
Fuente: (Gupta, 2009)

Las subrutinas descritas en laTabla 33 están definidas en el archivo "usb_host_cdc_interface.c" que pertenece a la capa "*CDC-ClassClient Driver*" del *stack* USB utilizado.

4. Comunicación RS-232 con un dispositivo externo La comunicación serial RS-232 con un dispositivo externo fue implementada utilizando el módulo UART del PIC24FJ64GB002. Esto permite que cualquier dispositivo externo que cuente con un puerto de comunicación serial RS-232 pueda controlar el radar láser e integrarlo a una aplicación en específico sin necesidad de que una computadora de tamaño real funcione como el anfitrión USB.

La implementación requiere que el dispositivo externo transmita la cadena de caracteres "GO/n" hacia el microcontrolador para solicitar que el radar láser inicie la adquisición de datos de distancia.

Figura 66: Algoritmo de recepción de datos en el microcontrolador

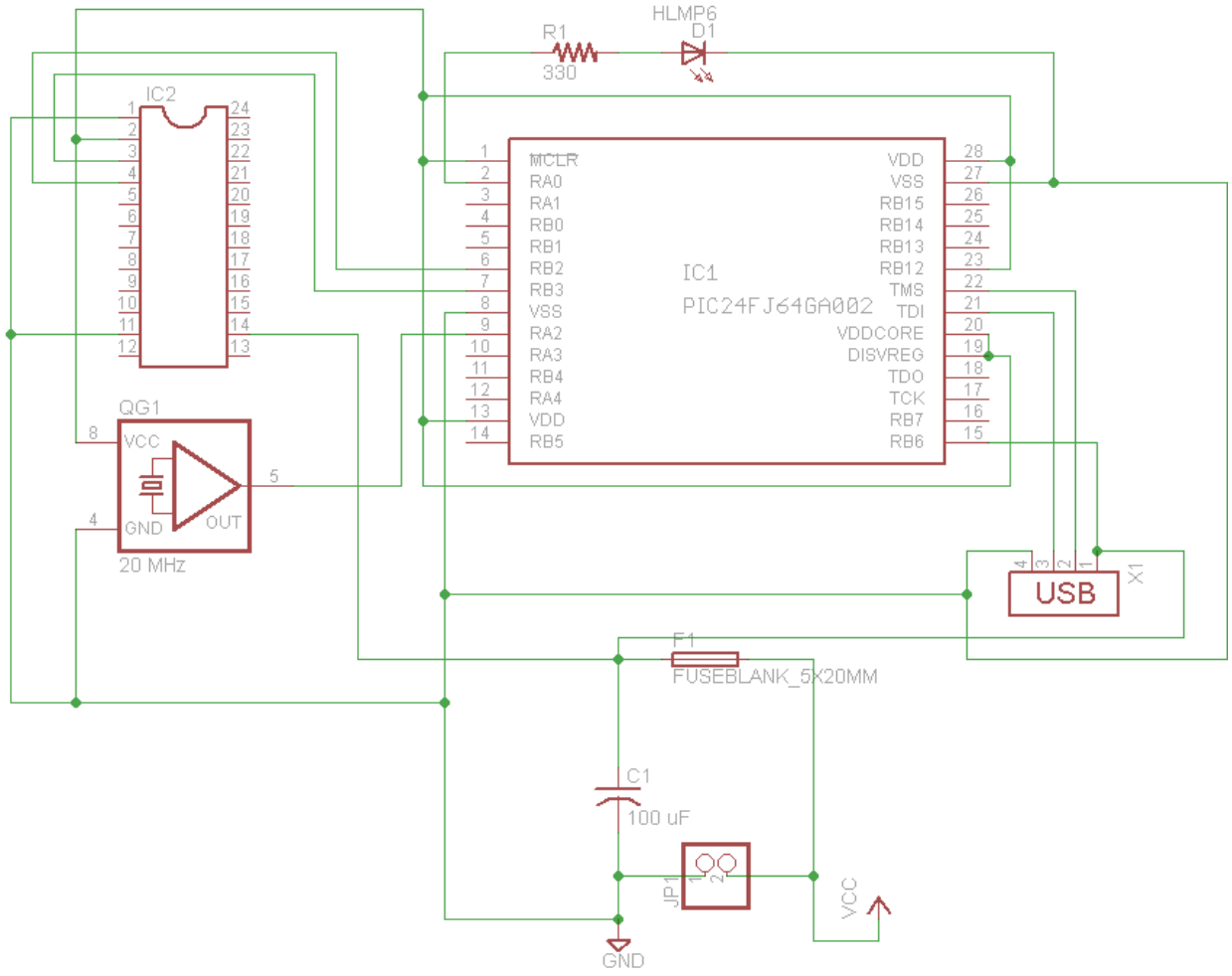


La Figura 66 muestra la estructura del algoritmo de recepción de datos implementado en el microcontrolador para detectar la recepción de la cadena de caracteres “GO/n”. Cuando el microcontrolador recibe dicha cadena de caracteres, este transmite el comando de adquisición de datos correspondiente (i.e. “MS”, “MD”, “GS”, “GD”) al radar láser a través del bus USB 2.0.

5. Diseño de la circuitería eléctrica

a. Circuito eléctrico diseñado El circuito diseñado para la implementación del anfitrión USB dedicado en el microcontrolador PIC24FJ64GB002 se muestra en la Figura 67.

Figura 67: Circuito diseñado para implementación de anfitrión USB



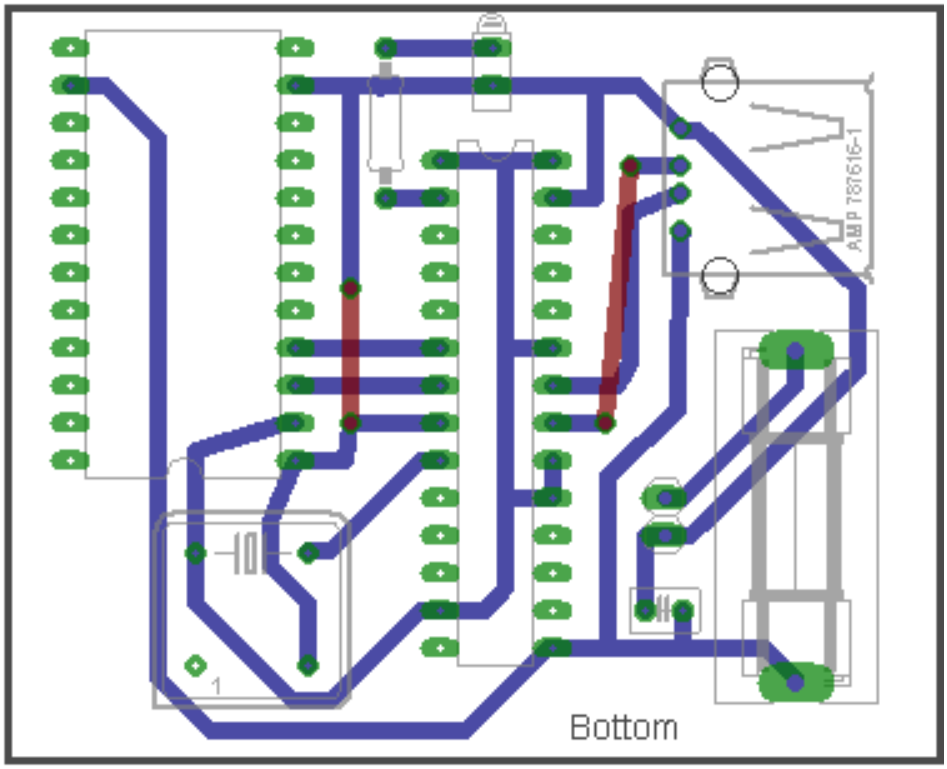
El circuito fue diseñado en el software EAGLE de CadSoft v6.2.0. La alimentación del circuito es de 5 VDC @ 1A (JP1) y cuenta con un fusible de protección de 1 amperio para proteger a los componentes del circuito y al sensor láser contra una sobre carga de corriente. Para ello, se utilizó un fusible tipo europeo de 22.5 mm.

La línea de alimentación de 5 VDC cuenta con un capacitor de 100 μF para eliminar variaciones de voltaje. Esta alimenta la placa adaptadora XBee 5V/3.3V (IC2) y la línea VBUS del receptáculo USB (X1-1) en donde es conectado el sensor láser URG-04LX-UG01. La placa adaptadora XBee 5V/3.3V posee un regulador de voltaje integrado de 3.3 VDC que alimenta al módulo de radio frecuencia XBee Pro de 60 mW y también provee una salida regulada de 3.3 VDC en su pin No. 2 que es utilizada para alimentar al PIC24FJ64GB002 y al oscilador EC de 20 MHz.

El *LED* (D1) conectado al pin RA0 del microcontrolador indica la recepción de un nuevo comando desde un dispositivo externo a través del protocolo RS-232. Este cambia de estado (*toogle*) cada vez que se genera la interrupción de recepción (Rx) en el módulo UART del microcontrolador.

b. Diseño de la placa impresa (PCB) La PCB diseñada para la implementación del anfitrión USB dedicado en el microcontrolador PIC24FJ64GB002 se muestra en la Figura 68.

Figura 68: Diseño de PCB con componentes para anfitrión USB dedicado



El diseño de la PCB fue realizado utilizando el software EAGLE de CadSoft v6.2.0. La Figura 68 muestra el diagrama de componentes del circuito impreso diseñado. Este cuenta con 2 capas; en la capa superior fue necesario ubicar 2 pistas (color rojo) por limitantes de espacio y el resto de pistas se ubican en la capa inferior (color azul).

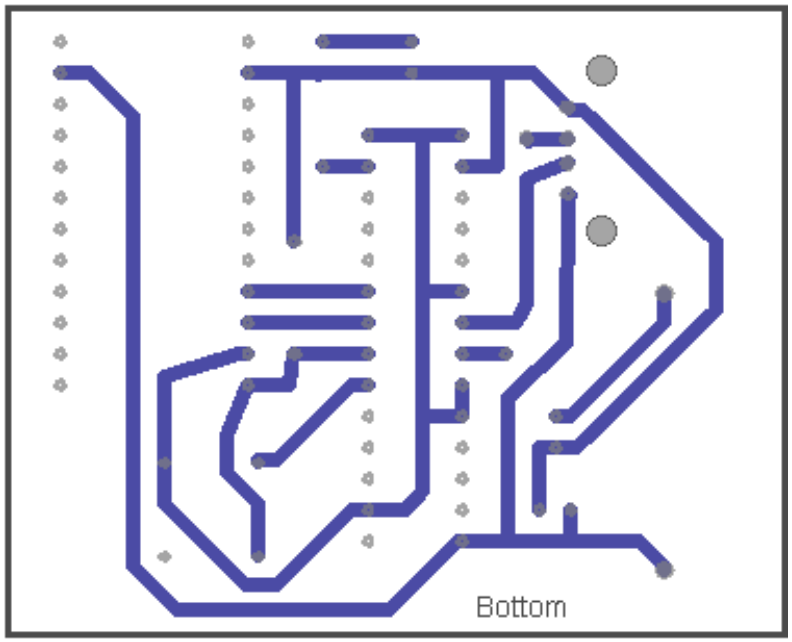
Los componentes utilizados para la fabricación del circuito impreso (PCB)se muestran en la Tabla 34.

Tabla 34: Componentes utilizados para la fabricación del circuito impreso

Cantidad	Descripción
1	Microcontrolador PIC24FJ64GB002 (SPDIP - 28 pines)
1	Socket DIP de 28 pines
1	Fusible de 1A (tipo europeo - 22.5 mm)
1	Porta fusible (tipo europeo - 22.5 mm)
40	*Headers* hembra
2	*Headers* macho
1	Capacitor 100µF (16 V)
1	Oscilador EC de 20 MHz
1	Receptáculo USB tipo Standard-A para PCB
1	LED verde con una resistencia de 330 Ω (1/4 W)
1	Módulo de radiofrecuencia *XBee* Pro de 60 mW
1	Placa adaptadora *XBee* 5V/3.3V (Parallax - #32401)

La Figura 69 muestra el diseño de las pistas de la capa inferior de la placa impresa utilizado para fabricarla empleando termo transferencia. Las conexiones de las 2 pistas de la capa superior de la placa fueron realizadas utilizando alambre de cobre.

Figura 69: Capa inferior del circuito impreso diseñado



A continuación se muestran fotografías del circuito impreso terminado.

Figura 70: PCB terminada – a) vista superior (izq.) b) vista inferior (der.)

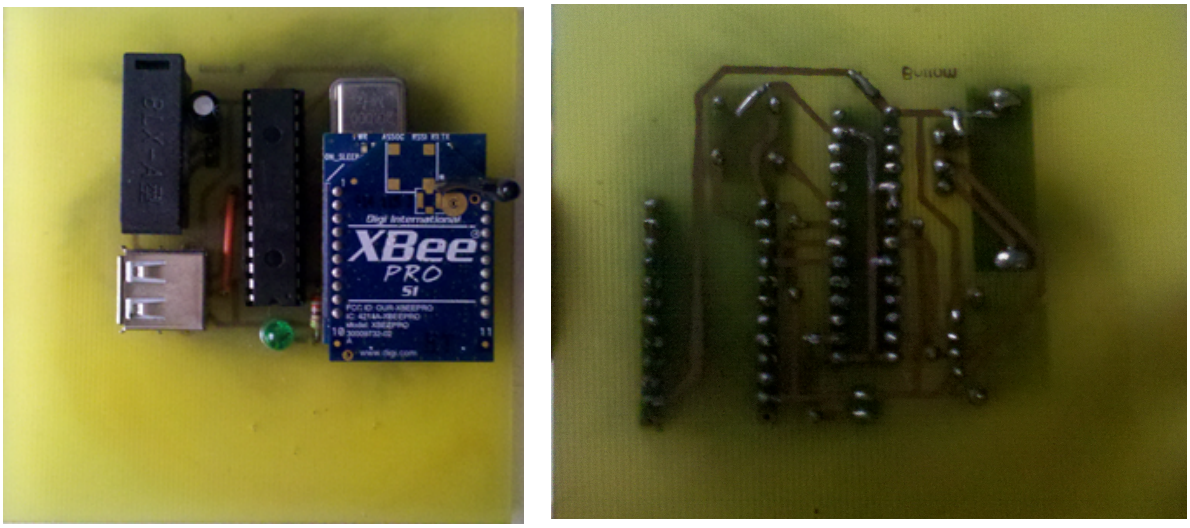
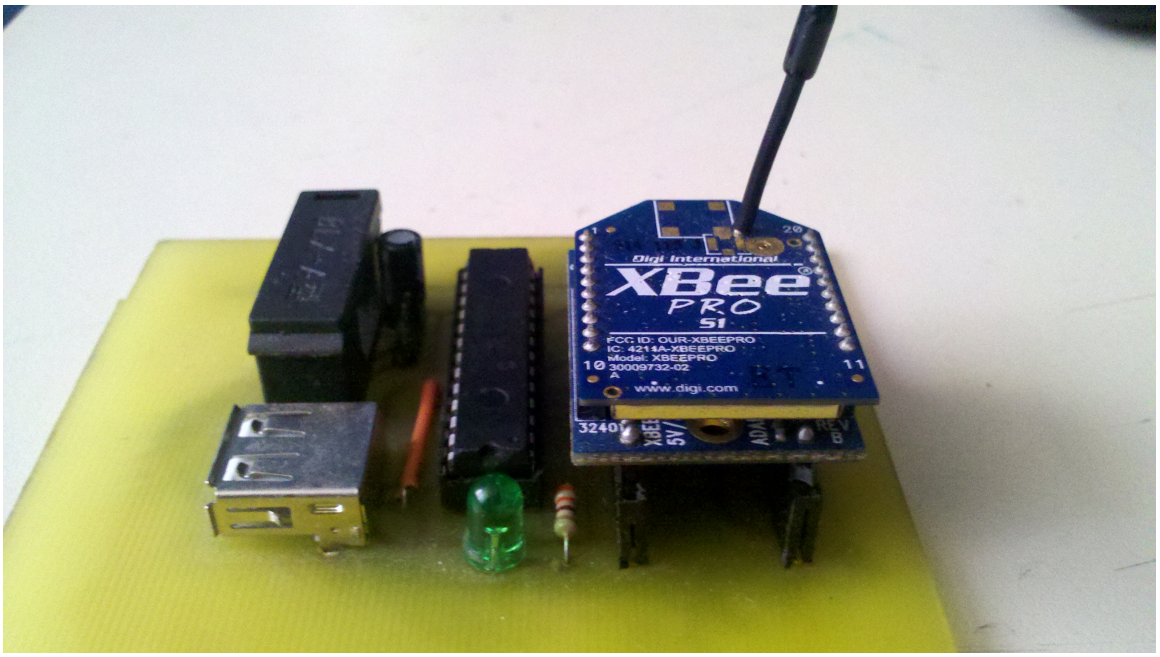


Figura 71: PCB terminada - vista frontal



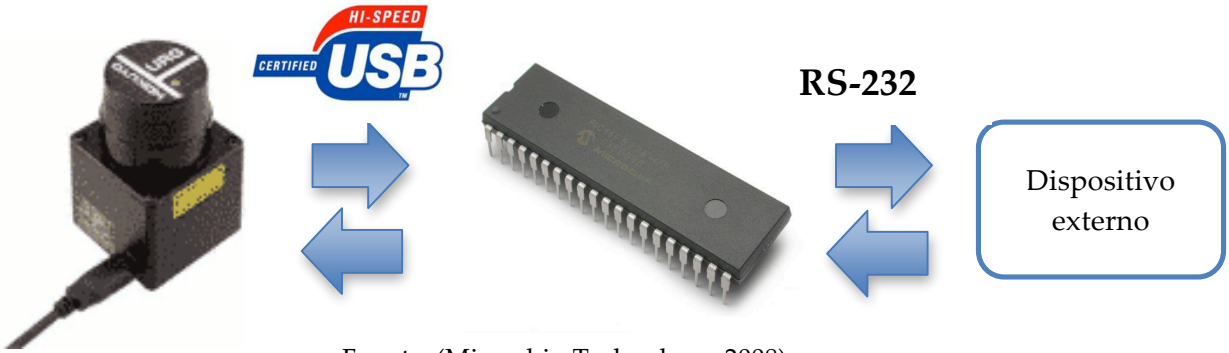
D. Funcionamiento y operación del controlador CDC

Como se mencionó anteriormente, la cadena de caracteres "GO\n" debe ser transmitida desde un dispositivo externo hacia el microcontrolador que implementa el anfitrión USB dedicado a través del protocolo RS-232. Dicha cadena serial de bytes se recibe en la terminal *"Rx"* del módulo UART del PIC24FJ64GB002.

Cuando el microcontrolador recibe la cadena "GO\n" este transmite el comando de adquisición de datos "MS" hacia el sensor láser a través del bus USB 2.0 y se encarga de decodificar y procesar la respuesta del sensor. Las subrutinas utilizadas por el anfitrión USB para comunicarse con el sensor láser a través de su interfaz USB 2.0 están definidas en la capa "*CDC-Class Client Driver*" del *stack* USB. Además, los parámetros del comando "MS" transmitido al sensor láser pueden configurarse gráficamente para una aplicación en específico utilizando la herramienta "*Simple-URG Viewer*"[4]. Dicha configuración es almacenada en la memoria EEPROM del microcontrolador.

Por último, los datos de distancia decodificados (en milímetros) contenidos en la respuesta del sensor son transmitidos hacia el dispositivo externo que los solicitó a través de la terminal *"Tx"* del módulo UART del microcontrolador. La Figura 72 ilustra el proceso descrito anteriormente.

Figura 72: Operación del controlador USB – CDC



Fuente: (Microchip Technology, 2008)

[4] Herramienta disponible en línea: www.simpleviewer.weebly.com

Por otro lado, el protocolo SCIP2.0 define 2 comandos de adquisición de datos múltiple. Estos se diferencian por su rango máximo, por el número de bytes que utilizan para codificar cada medición de distancia y por el número máximo de bytes que podrían llegar a transmitir (véase Tabla 35).

Tabla 35: Comandos de adquisición múltiple de datos

Comando	Rango min/max	Bytes de codificación	No. máx. de bytes transmitidos
MS	20 - 4095 mm	2	1.36 kB
MD	20 – 5096 mm	3	2.04 kB

Fuente: (Hokuyo, 2006)

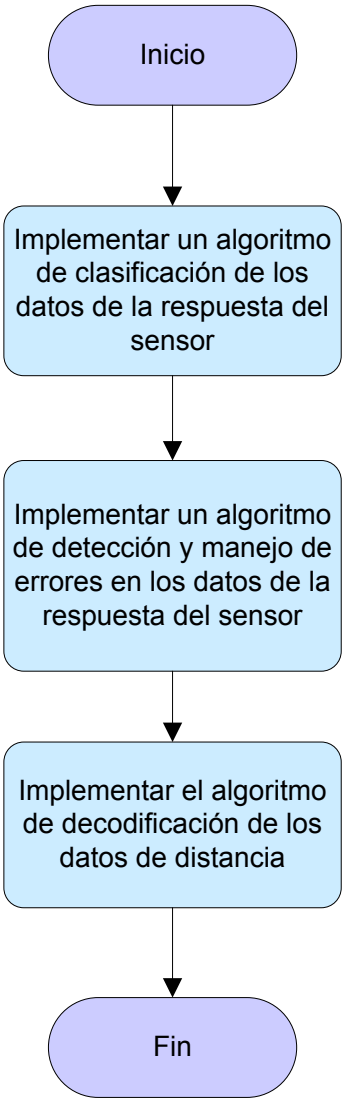
La interfaz de datos desarrollada en este trabajo de graduación implementa únicamente el comando de adquisición de datos múltiple "MS". La utilización del comando MS reduce significativamente la cantidad de datos transmitidos por el sensor (casi 0.7 kB menos) y el número de errores en la transmisión de los mismos.

La transmisión de grandes cantidades de datos del sensor ocasiona múltiples errores en el procesamiento de los datos ya que dicho procesamiento fue implementado en una micro arquitectura con recursos limitados de procesamiento y memoria como la de un microcontrolador.

VIII. PROCESAMIENTO DE LOS DATOS DE DISTANCIA

En este capítulo se describe el procesamiento de los datos de distancia del sensor láser; este incluye tres etapas: clasificación, detección y manejo de errores y decodificación de distancias. El diagrama de flujo de la Figura 73 muestra el diseño de este experimento.

Figura 73: Diseño experimental etapa 2

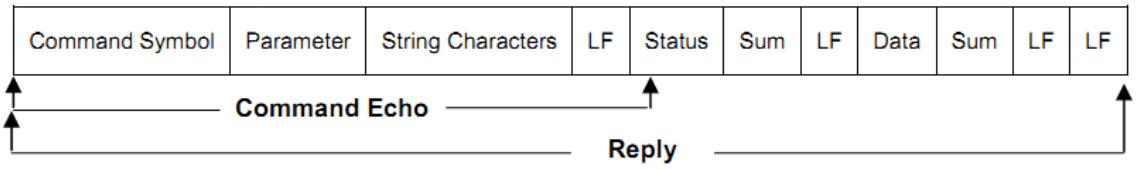


A. Clasificación de los datos provenientes del sensor láser

La primera etapa del procesamiento de los datos es la clasificación de los mismos. Los datos contenidos en la respuesta del sensor a un comando pueden ser concebidos como un grupo de datos seriales cuya longitud es variable y depende del comando transmitido al sensor.

En general, la respuesta del sensor URG-04LX-UG01 tiene la forma que se muestra en la Figura 74. El campo "*Data*" contiene la información enviada por el sensor en respuesta al último comando transmitido.

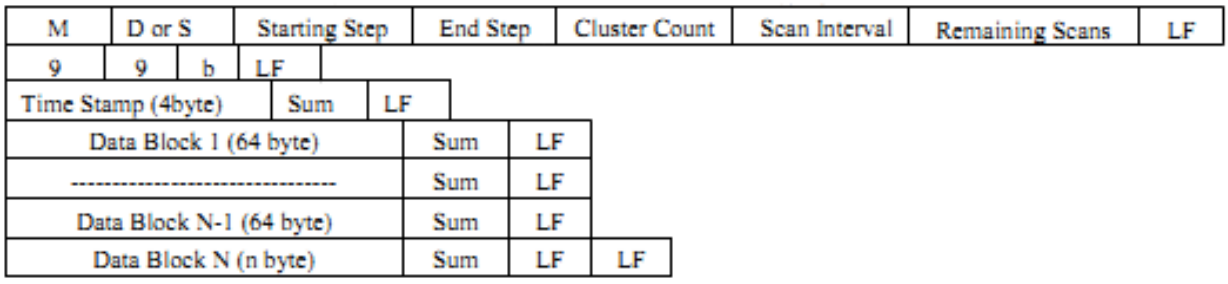
Figura 74: Formato de comunicación sensor-> Host



Fuente: (Hokuyo, 2006)

En los comandos de adquisición de datos, el campo "*Data*" contiene los datos de distancia y está compuesto por múltiples bloques de datos de hasta 65 bytes. Cada uno de estos bloques está compuesto por el bloque de datos o "Data Block" de hasta 64 bytes que contiene datos de distancia codificados y su byte de verificación "Sum" correspondiente (véase Figura 75).

Figura 75: Formato de comunicación en comandos de adquisición de datos



Fuente: (Hokuyo, 2006)

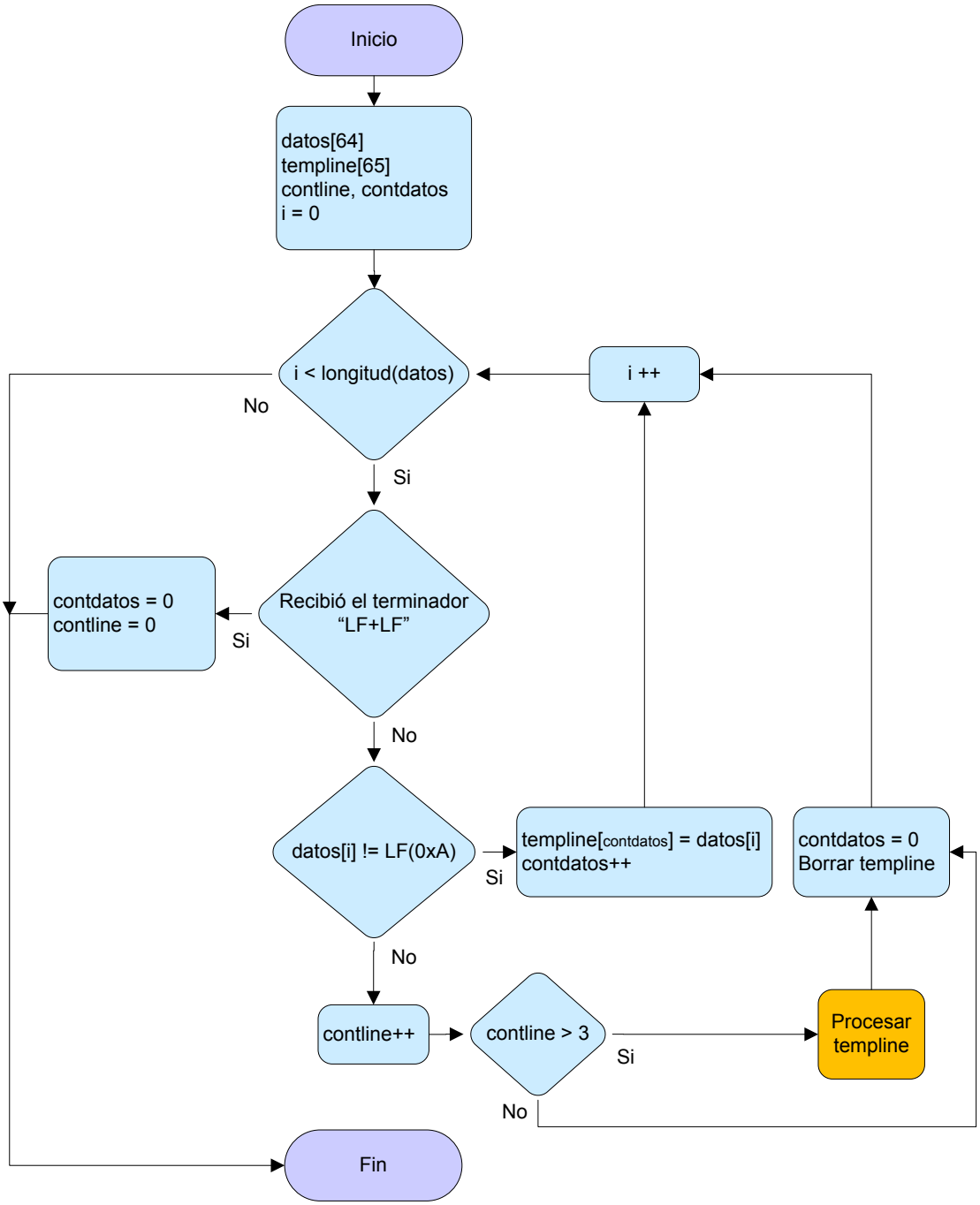
Por esta razón, surge la necesidad de implementar un algoritmo que clasifique la respuesta del sensor y permita separar todos los campos de datos contenidos en ella. Para implementar el algoritmo de clasificación fue necesario identificar un carácter que cumpliera la función de "separador" de los campos de datos en la respuesta del sensor.

En la Figura 75, el carácter LF (0xA) o "nueva línea" cumple esta función y permite separar la respuesta del sensor en líneas o bloques de datos de la siguiente forma:

- Línea No. 1 - Eco o repetición del comando transmitido (*Command Echo*)
- Línea No. 2 - Bytes de estado (*Status*)
- Línea No. 3 - Estampa de tiempo (*Timestamp*) *OPCIONAL
- A partir de la línea No. 3 o No. 4 - Datos (*Data*)
- Última línea - Terminador "LF"

El diagrama de flujo de la Figura 76 muestra, en pseudocódigo, la estructura del algoritmo de clasificación de datos implementado.

Figura 76: Algoritmo de clasificación de los datos



La Tabla 36 presenta una breve descripción de las variables utilizadas en el algoritmo de clasificación. El algoritmo fue implementado en el microcontrolador en la subrutina *"analizar"* (véase código fuente en el Apéndice).

Tabla 36: Variables del algoritmo de clasificación

Símbolo	Descripción
datos[ ]	Arreglo de datos actuales, su longitud máxima es 64 bytes y puede contener líneas completas o fragmentos de líneas de datos.
templine[ ]	Bloque o línea actual de datos, puede tener hasta 65 bytes de longitud (Data Block + Sum).
contline	Contador que refleja el número de líneas de datos contenidas en la respuesta del sensor.
contdatos	Contador que refleja el número de bytes en la línea actual de datos.
TxError	Bandera booleana que indica un error en los datos del bloque actual.
i	Índice del ciclo que recorre los arreglos de datos nuevos.

Entonces, el algoritmo de clasificación recorre los arreglos de datos nuevos en busca del carácter separador LF (0xA). Mientras no lo encuentre, añade el byte actual al final de la línea actual de datos (*templine*) e incrementa el contador de datos (*contdatos*). Cuando encuentra el carácter LF (0xA), incrementa el contador de líneas (*contline*) y, si la línea actual pertenece al campo *Data* (*contline*>3), entonces la procesa. Es decir, el algoritmo procesa únicamente los datos del campo "*Data*"; estos están ubicados a partir de la línea No. 4 de datos.

B. Detección y manejo de errores en los datos provenientes del sensor láser

Según el diseño de este experimento, la segunda etapa del procesamiento de los datos es la detección y manejo de errores en la transmisión de los mismos. Los datos contenidos en la respuesta del sensor pueden contener errores debido a distintos motivos. Estos errores podrían representar la presencia de caracteres incorrectos o un número incorrecto de caracteres en la respuesta.

1. Detección de errores La detección de errores en la respuesta del sensor se llevó a cabo utilizando el byte de autenticación descrito en la sección *"Errores en la respuesta del sensor"* pág. 60, llamado SUM. La detección de errores fue implementada únicamente para los comandos de adquisición de datos (i.e. MS, MD, GS, GD) debido a que la respuesta del sensor a este tipo de comandos contiene grandes cantidades de datos de distancia. Además, es de vital importancia que estos datos no contengan errores ya que podrían proveer información incorrecta acerca del entorno del robot. El campo *"Data"* de la respuesta del sensor a los comandos de adquisición de datos tiene la forma mostrada en la Figura 77.

Figura 77: Formato del campo "Data" de la respuesta a comandos de adquisición

Data Block 1 (64 byte)	Sum	LF	
----------	Sum	LF	
Data Block N-1 (64 byte)	Sum	LF	
Data Block N (n byte)	Sum	LF	LF

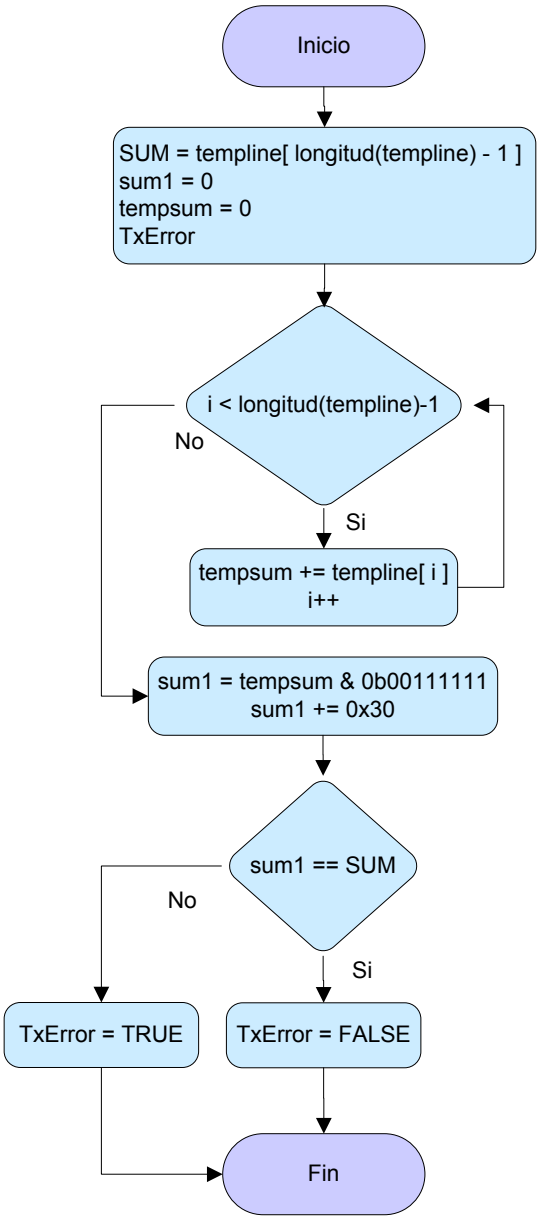
Fuente: (Hokuyo, 2006)

El algoritmo de clasificación descrito en la sección anterior permite separar los datos del campo "Data" en bloques de hasta 65 bytes para ser analizados que contienen el bloque de datos de distancia o "Data Block" (hasta 64 bytes) y su byte de verificación "Sum" correspondiente (1 byte).

Los bloques de datos hasta 64 bytes requieren ser analizados para determinar si se produjo algún error en la transmisión. Por esta razón, surgió la necesidad de implementar un algoritmo de detección de errores que detecte errores en los datos de distancia de cada bloque utilizando el byte de verificación "Sum" correspondiente.

El diagrama de flujo de la Figura 78 muestra, en pseudocódigo, la estructura del algoritmo de detección de errores implementado.

Figura 78: Algoritmo de detección de errores en los datos de distancia



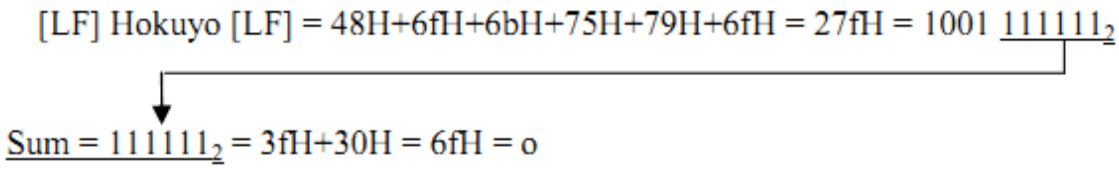
La Tabla 37 presenta una breve descripción de las variables utilizadas en el algoritmo de detección de errores. El algoritmo de detección fue implementado en el microcontrolador en la subrutina *"checkSUM"* que forma parte de la subrutina *"procesar"* que se encarga del manejo de errores en los datos (véase código fuente en el Apéndice).

Tabla 37: Variables del algoritmo de detección de errores

Símbolo	Descripción
templine[ ]	Bloque o línea actual de datos, puede tener hasta 65 bytes de longitud (Data Block + Sum).
SUM	Byte de verificación del bloque actual de datos transmitido por el sensor; es el último elemento del arreglo templine[].
sum1	Byte de verificación del bloque actual de datos calculado con los datos recibidos.
tempsum	Suma aritmética de los códigos ASCII de cada uno de los caracteres en el bloque actual de datos.

Entonces, el algoritmo de detección de errores calcula el byte de verificación del bloque actual de datos(*sum1*)de la siguiente manera: Primero, suma algebraicamente todos los códigos ASCII de los datos del bloque como lo muestra la Figura 79; el resultado de la suma se almacena en *"tempsum"*.

Figura 79: Autenticación de los datos en la respuesta del sensor



Fuente: (Hokuyo, 2006)

Finalmente, se toman los 6 bits menos significativos del resultado *tempsum* y se añade 48 (0x30) a este valor. El resultado obtenido es el byte de verificación del bloque actual de datos calculado utilizando los datos recibidos y se almacena en *"sum1"*.

Por último, el algoritmo de detección compara el byte de verificación calculado *"sum1"* con el byte de verificación transmitido por el sensor *"SUM"*(ubicado en la última posición del bloque actual de datos). Si ambos bytes de verificación coinciden, entonces el bloque actual no contiene errores (TxError = FALSE). De lo contario, el bloque actual de datos de distancia contiene errores (TxError = TRUE).

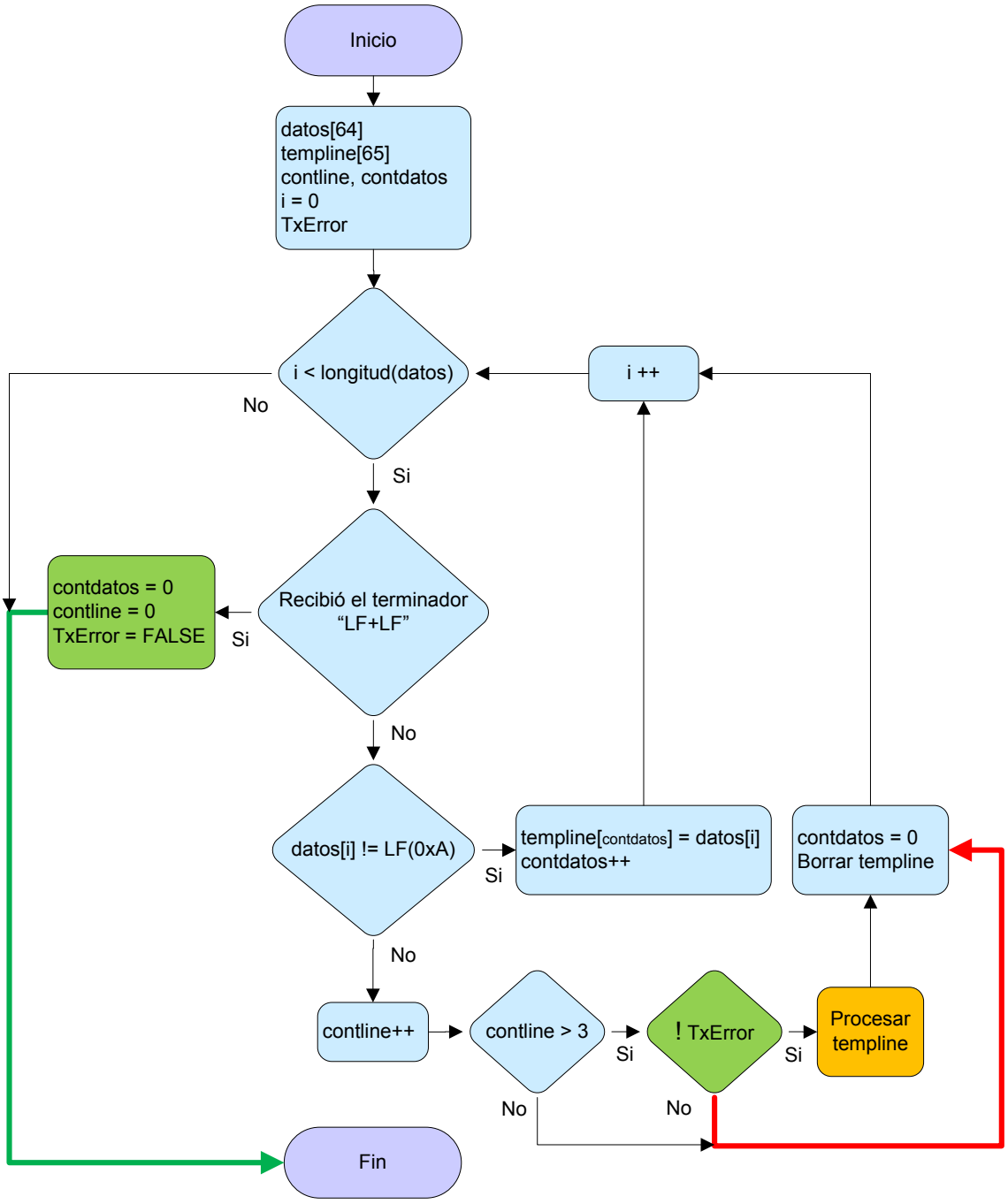
2. Manejo de errores Según el diseño experimental de la Figura 73, se requería implementar un algoritmo de manejo de errores que se encargara de manejar los errores en los datos de distancia de la respuesta del sensor.

Como se mencionó anteriormente, la importancia del manejo de los errores en la transmisión radica en que los datos de distancia erróneos podrían proveer información incorrecta acerca del entorno del robot al ser decodificados. Esta información incorrecta acerca del entorno ocasionaría que el sistema de control del robot tomara decisiones equivocadas que podrían poner en riesgo la integridad física del robot.

Por lo anterior, surgió la necesidad de implementar un algoritmo de manejo de errores que manejara los errores detectados por el algoritmo de detección descrito en la sección anterior. Dicho algoritmo de detección utiliza la bandera *TxError* para reflejar, en todo momento, el resultado de la detección de errores del último bloque de datos de distancia recibido.

El algoritmo de manejo de errores fue implementado en el microcontrolador en la subrutina *"procesar"* (véase código fuente en el Apéndice). El diagrama de flujo de la Figura 80muestra, en pseudocódigo, el algoritmo de manejo de errores integrado al algoritmo de clasificación descrito anteriormente.

Figura 80: Algoritmo de manejo de errores



El algoritmo de manejo de errores de la Figura 80 establece que si el bloque actual de datos pertenece al campo "Data" (contline>3) y no se detectan errores en los datos de distancia de dicho bloque (TxError = FALSE) entonces estos son decodificados en el bloque "*Procesar templine*".

El bloque *"procesar templine"* en el diagrama de flujo de la Figura 80 está implementado específicamente en la subrutina *"getDistancias"* del microcontrolador. Este proceso es repetido hasta que todos los bloques de datos de distancia del campo "Data" de la respuesta del sensor hayan sido decodificados.

De lo contrario, si se detectan errores en el bloque actual de datos (TxError = TRUE), la decodificación de datos es deshabilitada temporalmente (véase trazo rojo en la Figura 80). Si esto ocurre, los datos del bloque actual y todos los bloques de datos de distancia posteriores de la respuesta del sensor no son decodificados y se pierden. En otras palabras, se ignoran todos los datos de distancia del último barrido de medición del radar láser. De esa manera se evita la decodificación de datos erróneos que podrían generar información incorrecta acerca del entorno del robot.

Sin embargo, la decodificación de los datos es habilitada nuevamente cuando el algoritmo de clasificación detecta que se ha recibido el terminador "LF + LF"; esto indica el final de la respuesta actual del sensor con errores (véase trazo verde en la Figura 80). Posteriormente, el algoritmo está listo para clasificar y procesar los datos de distancia de un nuevo barrido de medición.

C. Decodificación de los datos de distancia del sensor láser

La última etapa del procesamiento de los datos es la decodificación de los datos de distancia. La información proveniente del sensor láser está codificada con el propósito de reducir el tiempo de transmisión.

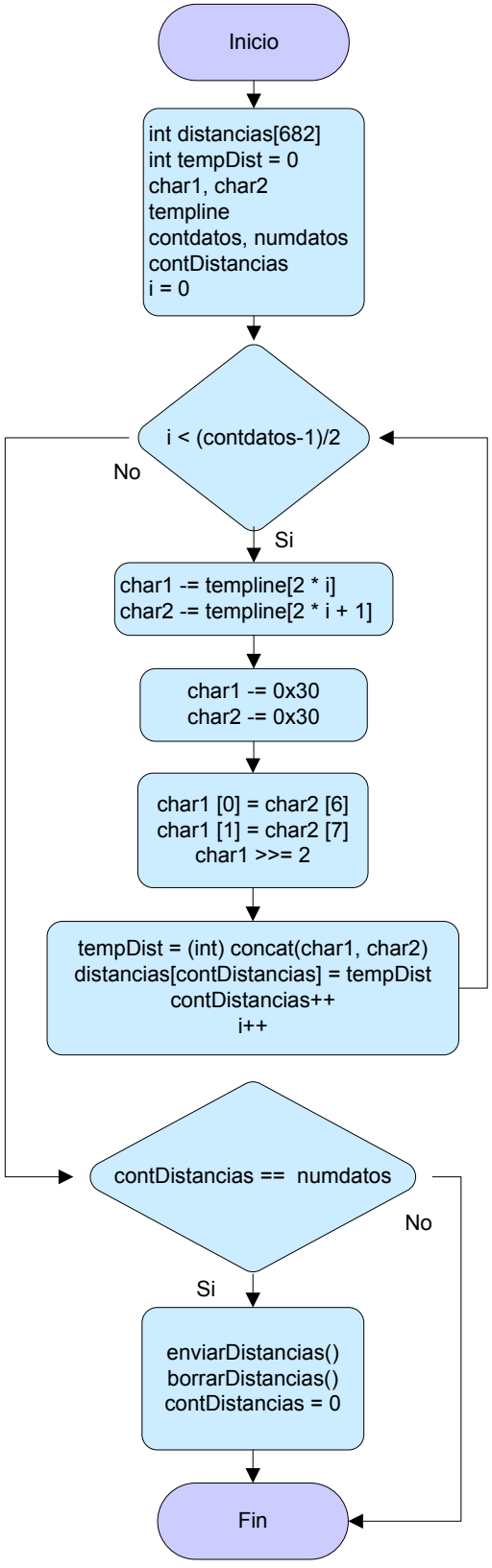
Los datos de distancia de la respuesta del sensor a un comando de adquisición de datos están codificados y contienen las mediciones de distancia efectuadas por el sensor láser en cada uno de los "pasos" o puntos de medición dentro del rango de detección. Dichos datos pueden ser decodificados para obtener las distancias, en milímetros, medidas por el sensor láser en un barrido de medición.

Por lo anterior, surgió la necesidad de implementar un algoritmo de decodificación de datos que decodificara los datos de distancia contenidos en la respuesta del sensor y permitiera obtener las distancias en milímetros medidas por el sensor a lo largo del rango de detección (véase Figura 81).

Figura 81: Decodificación de los datos de distancia

Se utilizó el algoritmo de decodificación para datos de "2 caracteres" detallado en la especificación del protocolo de comunicación del sensor, el SCIP2.0. Este tipo de codificación utiliza 2 caracteres ASCII para representar 1 dato de distancia con una longitud máxima de 6*2 = 12 bits en un rango comprendido entre 0 y 4,095 mm (2^12 – 1 = 4095). El diagrama de flujo de la Figura 82 muestra la estructura del algoritmo de decodificación; este representa la última etapa del procesamiento de los datos.

Figura 82: Algoritmo de decodificación de los datos de distancia



El algoritmo de decodificación de datos fue implementado en el microcontrolador en la subrutina *"decodechar"* que forma parte de la subrutina *"getDistancias"* que se encarga de obtener y almacenar las distancias en milímetros del último barrido de medición (véase código fuente en el Apéndice). La Tabla 38 presenta una breve descripción de las variables utilizadas en el algoritmo de decodificación de datos.

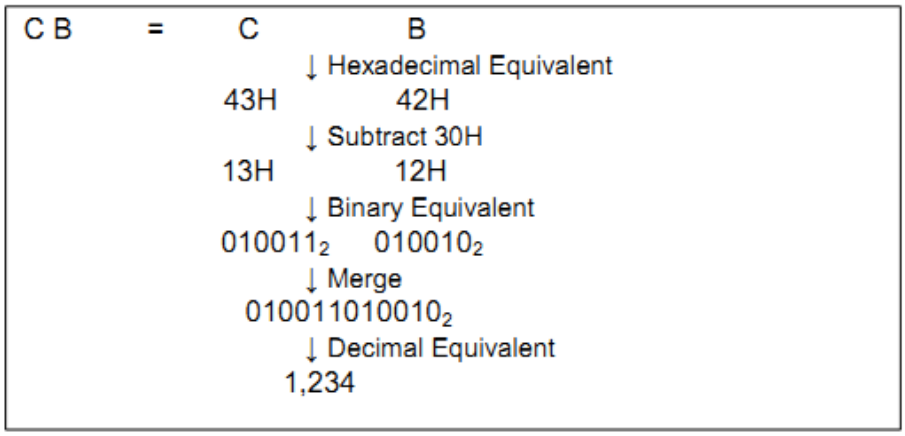
Tabla 38: Variables del algoritmo de decodificación de datos de distancia

Símbolo	Descripción
distancias[ ]	Arreglo donde se almacenan las distancias (mm) del último barrido de medición.
contDistancias	Contador de distancias decodificadas y almacenadas en el arreglo distancias[ ]
tempDist	Almacena la última distancia (mm) decodificada a partir de char1, char2.
char1, char2	Grupo de 2 bytes codificados en ASCII que contienen una medición de distancia (mm).
templine[ ]	Bloque o línea actual de datos, puede tener hasta 65 bytes de longitud (Data Block + Sum).
contdatos	Contador que refleja el número de bytes en la línea actual de datos *templine[]*.
numdatos	Número total de distancias en un barrido de medición calculado como: (paso final – paso inicial)/clustercount
i	Índice del ciclo que recorre la línea actual de datos.

El algoritmo de decodificación de datos recorre la línea actual de datos *"templine"* y en cada iteración almacena 2 bytes de datos adyacentes en las variables char1 y char2 respectivamente. Luego, decodifica ambos bytes para obtener una distancia medida en milímetros y la almacena en la variable *"tempDist"*.

Para decodificar el dato de distancia contenido en las variables *"char1"* y *"char2"* es necesario restarle a ambas la cantidad 0x30, luego se forma un nuevo valor concatenando los 6 bits menos significativos de *"char1"* y *"char2"* de la siguiente forma: char1<6:0> + char2<6:0>. Finalmente, el valor formado al concatenar dichos campos representa la distancia decodificada en milímetros (véase Figura 83). Esta distancia tiene 12 bits de longitud (2*6 bits c/u) y se almacena en la variable *"tempDist"*.

Figura 83: Decodificación de datos de distancia de 2 bytes



Fuente: (Hokuyo, 2006)

Posteriormente, la distancia decodificada es almacenada en una posición del arreglo *"distancias"* y después se incrementa el contador *"contDistancias"*. Luego, se procede a decodificar el siguiente grupo adyacente de 2 bytes de la línea actual de datos *"templine"* y el proceso se repite sucesivamente hasta llegar al final de la línea actual.

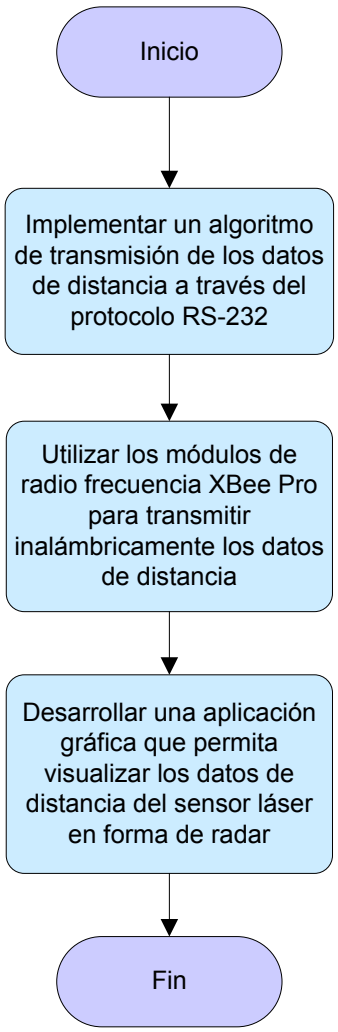
A continuación, se almacena la siguiente línea de datos de la respuesta del sensor en el arreglo *"templine"* y se repite el proceso de decodificación. Cuando se hayan decodificado todas las líneas de datos contenidas en la respuesta del sensor (*contDistancias == numdatos*) estas son transmitidas hacia un dispositivo externo a través del protocolo RS-232 utilizando la subrutina *"enviarDistancias"*.

Al finalizar la transmisión inalámbrica de las distancias, se borra el contenido del arreglo *"distancias"* para dar espacio a las distancias de un nuevo barrido de medición y se reinicia el contador de distancias decodificadas (*contDistancias = 0*).

IX. TRANSMISIÓN INALÁMBRICA Y VISUALIZACIÓN DE LOS DATOS DE DISTANCIA

En este capítulo se describe detalladamente la transmisión inalámbrica y la visualización de los datos de distancia del sensor láser. El diagrama de flujo de la Figura 84 muestra el diseño de este experimento.

Figura 84: Diseño experimental etapa 3

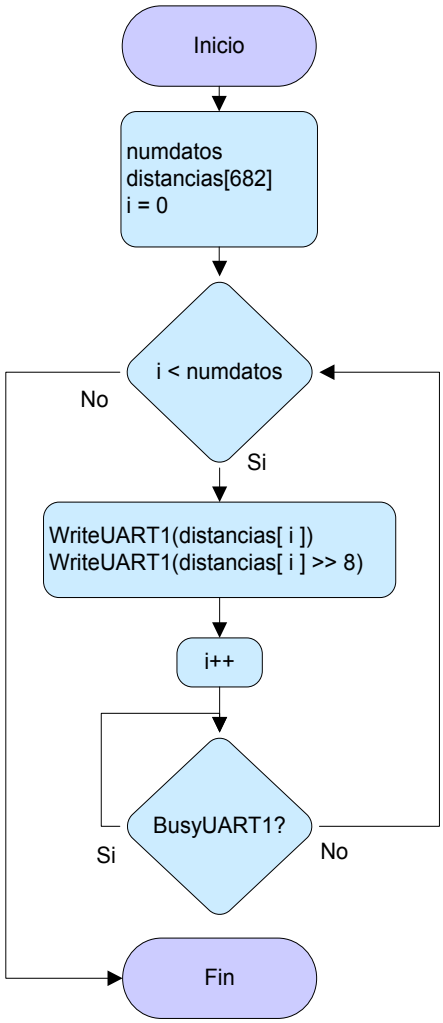


A. Transmisión de los datos desde el microcontrolador

Cuando el algoritmo de decodificación de datos descrito en la sección anterior ha decodificado todos los datos de distancia del último barrido de medición (contDistancias = numdatos) es necesario transmitir las distancias almacenadas en el arreglo *"distancias"* a través del protocolo RS-232 para actualizar la información acerca del entorno del robot en el sistema de reconocimiento de entorno.

Por esta razón, según el diseño del experimento de la Figura 84, se implementó un algoritmo de transmisión de datos capaz de transmitir los datos de distancia contenidos en el arreglo *"distancias"*.

Figura 85: Algoritmo de transmisión de datos



El diagrama de flujo de la Figura 85 muestra la estructura del algoritmo de transmisión de datos implementado.

El algoritmo de transmisión de datos fue implementado en el microcontrolador en la subrutina *"enviarDistancias"* (véase código fuente en el Apéndice). El algoritmo utiliza el módulo UART del microcontrolador PIC24FJ64GB002 para transmitir las distancias almacenadas en el arreglo *"distancias"*. Dicho arreglo almacena los datos de distancia decodificadas como números enteros sin signo de 16 bits (distancias[i] <15:0>).

El algoritmo de transmisión de datos recorre el arreglo *"distancias"* y en cada iteración transmite el byte menos significativo de la distancia actual (distancias[i] <7:0>) y luego el byte más significativo de la misma (distancias[i] <15:8>) utilizando la función "WriteUART1". Por último, el algoritmo se detiene cuando ha transmitido el número de distancias especificado en la variable *"numdatos"*.

Fue necesario transmitir cada dato de distancia de esta manera debido a que el módulo UART del microcontrolador fue configurado para transmitir datos de hasta 8 bits de longitud.

B. Transmisión inalámbrica de los datos de distancia

Para transmitir inalámbricamente los datos de distancia del sensor láser se utilizaron 2 módulos transmisores/receptores de radio frecuencia XBee Pro de 60 mW; dichos módulos poseen una interfaz de datos UART (véase Figura 86).

Los datos de distancias son transmitidos de forma serial desde el microcontrolador hacia el módulo de radio frecuencia XBee Pro ubicado en el circuito impreso a través del módulo UART del PIC24FJ64GB002.

El módulo XBee Pro de 60 mW ubicado en el circuito impreso es alimentado a través de una placa adaptadora XBee 5V/3.3V fabricada por *Parallax* (véase Figura 86).

Figura 86: Módulo XBee Pro de 60 mW

Fuente: (Acroname Robotics, 2011)

Los datos de distancia son recibidos por un segundo módulo XBee Pro de 60 mW conectado a una placa adaptadora XBee-USB fabricada por *Parallax* (véase Figura 86). Dicha placa adaptadora posee un receptáculo mini-USB que permite la conexión hacia un dispositivo externo; por ejemplo, una computadora.

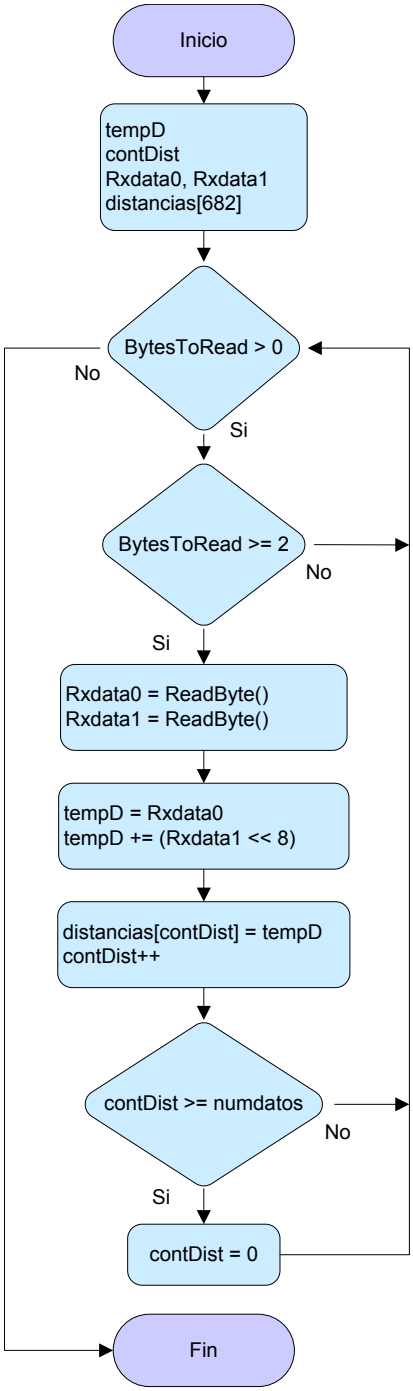
C. Visualización gráfica de los datos de distancia del radar láser

Según el diseño experimental de la Figura 84 era necesario desarrollar una aplicación gráfica en un lenguaje de alto nivel que permitiera visualizar gráficamente las mediciones de distancia efectuadas por el sensor láser en forma de un radar.

Por esta razón, se desarrolló una aplicación gráfica en el lenguaje de alto nivel C sharp (C#) para poder visualizar los datos del sensor en forma de radar. La aplicación desarrollada *"Simple-URG Viewer"* permite interactuar con los comandos definidos en el protocolo de comunicación SCIP2.0 a través de una consola y configurar los parámetros de funcionamiento del radar láser gráficamente. Además, se implementó un algoritmo de recepción de datos capaz de recibir los datos de distancia provenientes del microcontrolador.

La Figura 87 muestra la estructura del algoritmo de recepción de datos implementado.

Figura 87: Algoritmo de recepción de datos



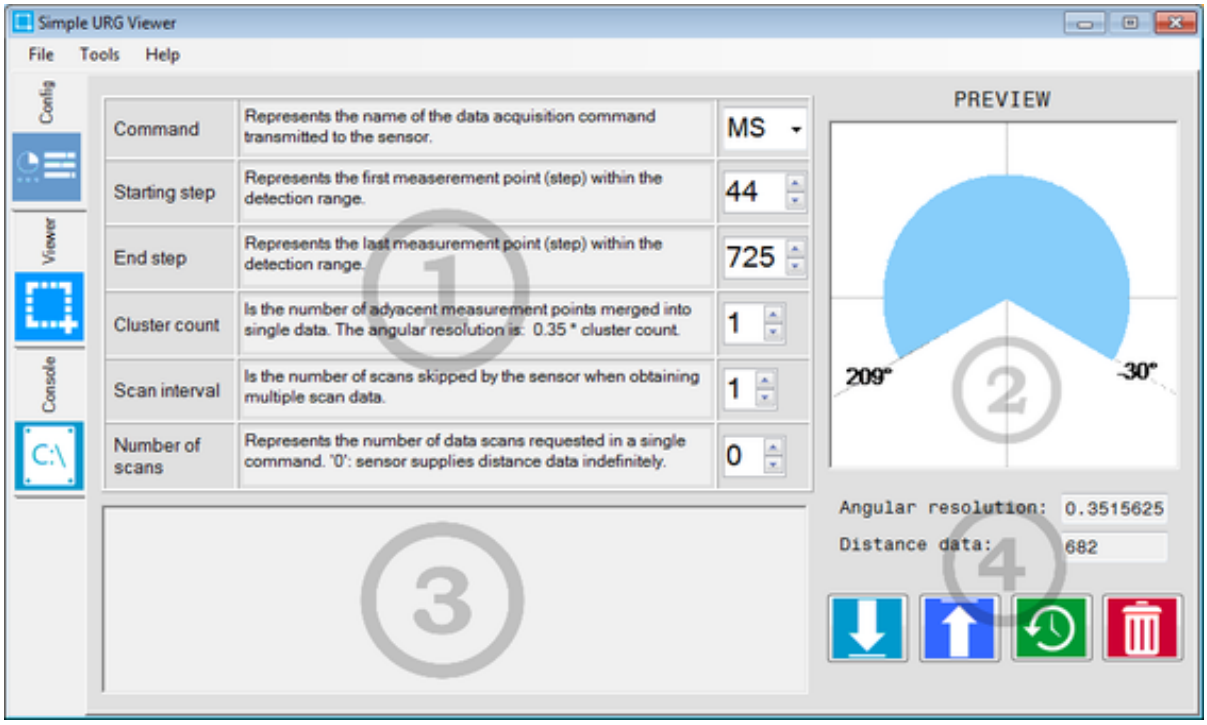
El algoritmo de la Figura 87 fue implementado en la aplicación gráfica desarrollada en C# en la subrutinas “DoUpdate” y “analizarVIEWER”.

1. Configuración de los parámetros de funcionamiento. La aplicación gráfica desarrollada *"Simple-URG Viewer"* permite modificar los parámetros del comando de adquisición de datos "MS" transmitido al sensor y almacenar la configuración actual en la memoria EEPROM del microcontrolador. Los parámetros de configuración que pueden modificarse en la sección 1 son los siguientes:

- Resolución angular
- Frecuencia de captura
- Número de capturas o mediciones de distancia
- Paso inicial y final del rango de detección

La sección 2 de la pestaña *"Config"* de la aplicación muestra una vista previa del rango de detección del radar (celeste), la resolución angular actual y la cantidad de datos de distancia contenidos en un barrido de medición (véase Figura 88).

Figura 88: Pestaña de configuración del "Simple-URG Viewer"



La sección 3 de la pestaña *"Config"* la aplicación es un *"Log"* y muestra mensajes de notificación al usuario acerca del estado actual de la configuración de la aplicación. Dicha configuración se almacena en la memoria EEPROM del microcontrolador.

Por último, los controles de la sección 4 permiten leer y escribir en la memoria no volátil del microcontrolador (EEPROM) la configuración actual de los parámetros del comando de adquisición de datos MS (véase Tabla 39).

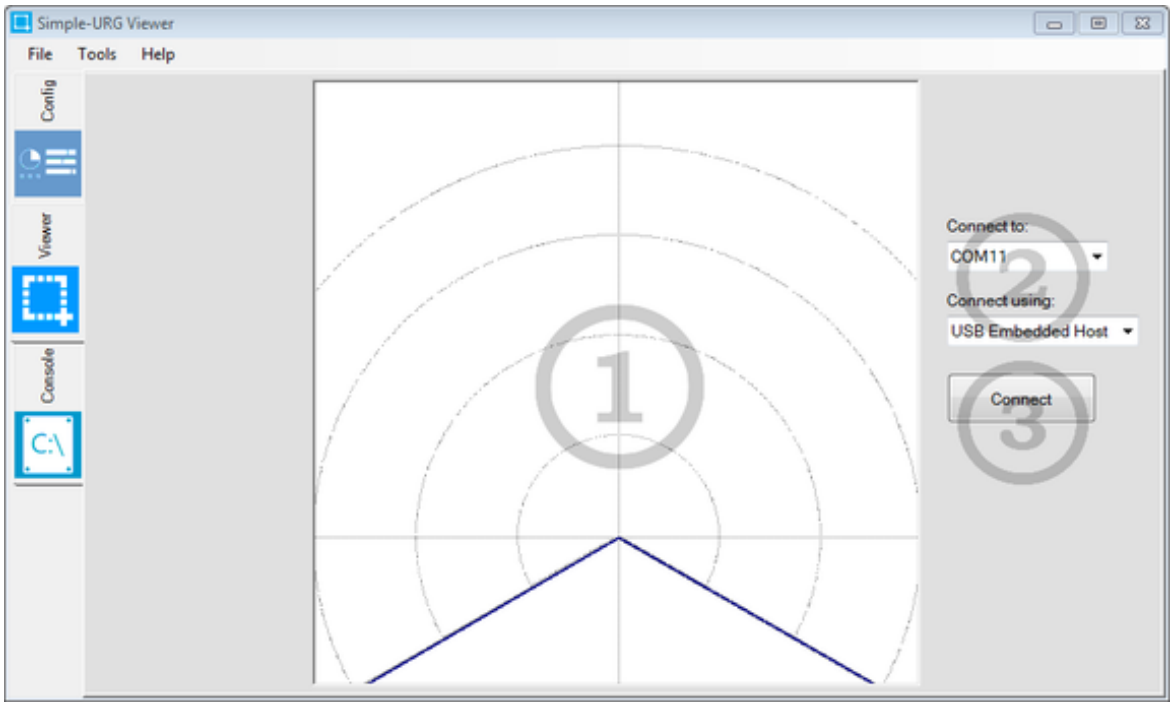
Tabla 39: Botones de la sección 4 en la pestaña "Config" del "Simple-URG Viewer"

Nombre	**Botón**	Descripción
Download		Escribe la configuración actual de los parámetros del comando "MS" de la aplicación gráfica en la memoria EEPROM del microcontrolador.
Upload		Lee la configuración de los parámetros del comando "MS" almacenada en la memoria EEPROM del microcontrolador y actualiza la aplicación gráfica.
Reset		Retorna los valores de los parámetros del comando "MS" a sus valores por defecto.
Clear Log		Borra el *"Log"* de eventos.

2. Visualización de los datos en forma de radar. La pestaña de visualización del *Simple-URG Viewer* denominada "Viewer" permite visualizar en tiempo real los datos de distancia del sensor láser en forma de radar bi-dimensional.

El radar despliega gráficamente el entorno del sensor y el rango de detección actual. Para ello, fue necesario realizar una conversión de los datos de distancia del sensor de coordenadas polares a cartesianas (véase Figura 89).

Figura 89: Pestaña de visualización del "Simple-URG Viewer"



- El control *"Connect to:"* permite seleccionar el puerto de comunicación serial a través del cual la aplicación se comunicará con el radar láser.

- El control *"Connect using:"* permite seleccionar el modo de conexión de la aplicación con el radar láser. Este puede ser: directamente a través de un puerto USB 2.0 disponible en una computadora o utilizando la interfaz de datos RS-232 implementada en el microcontrolador.

- El botón *"Connect"* conecta la aplicación al radar láser para iniciar la visualización de los datos de distancia en tiempo real.

3. Modo consola. La pestaña denominada "Console" permite visualizar la respuesta del radar láser a todos los comandos definidos en el protocolo de comunicación SCIP2.0 (véase Figura 90).

En este modo de operación se muestra la respuesta del sensor a los comandos trasmitidos sin realizar ningún tipo de procesamiento y/o decodificación previa de los datos recibidos.

Figura 90: Pestaña Consola del "Simple-URG Viewer"

- El botón *"Send"* transmite el comando del cuadro de texto 1 hacia el radar láser utilizando el modo de conexión seleccionado en la pestaña *"Viewer"*.

- El cuadro de texto 2 muestra la respuesta del sensor láser al comando transmitido.

- El botón "Clear" borra el contenido del cuadro de texto 2.

X. INTEGRACIÓN DE LA INTERFAZ DE DATOS A OTROS PROYECTOS

A. Megaproyecto SEEQ: Smart Environment Exploring Quadcopter

Esta aplicación requería un sensor capaz de proveer grandes cantidades de información acerca del entorno del cuadricóptero con alta resolución y exactitud. Además, era necesario que el sensor actualizara la información acerca del entorno varias veces por segundo debido a que el sistema de estabilización y vuelo debía funcionar en tiempo real. (Búrbano, Saravia, Sandoval, Suazo, & Morales, 2011)

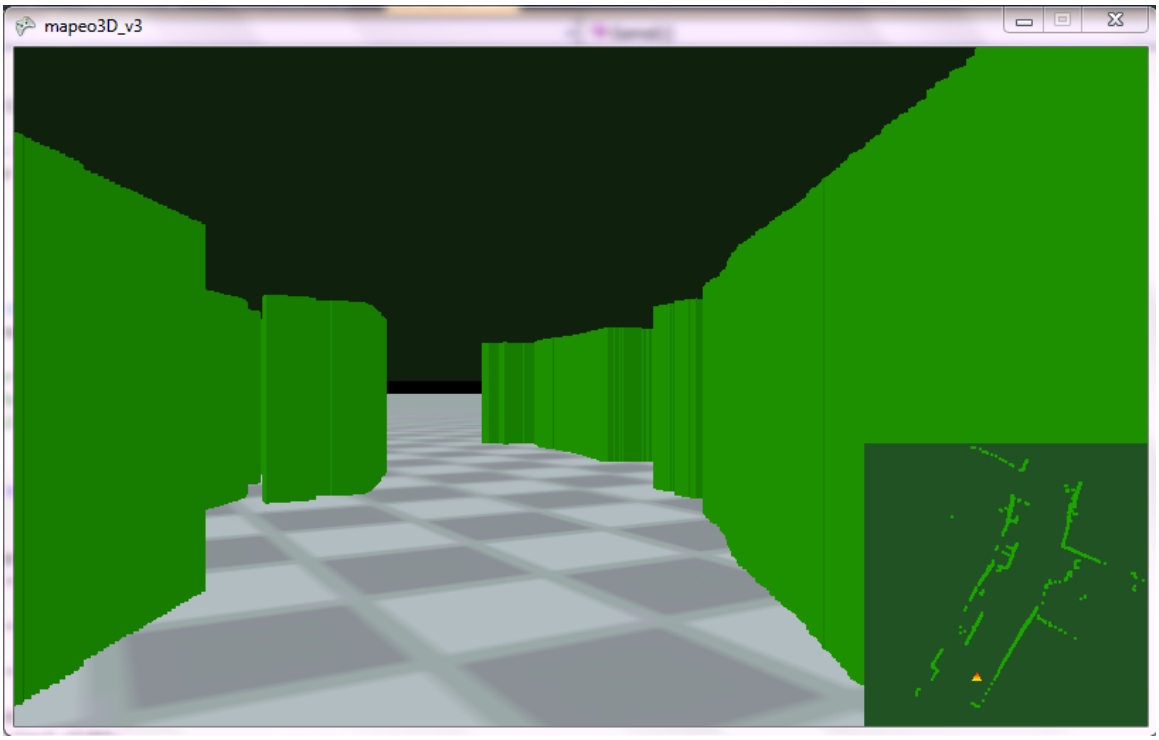
La Figura 91 muestra el sensor láser Hokuyo URG-04LX-UG01 integrado al cuadricóptero.

Figura 91: Sensor láser URG-04LX-UG01 integrado al cuadricóptero SEEQ

Fuente: (Búrbano, Saravia, Sandoval, Suazo, & Morales, 2011)

Por otro lado, la Figura 92 muestra la aplicación gráfica que permite visualizar los mapas tri-dimensionales del entorno del cuadricóptero generados a partir de los datos de distancia obtenidos con el sensor láser.

Figura 92: Aplicación gráfica de reconocimiento de entorno del megaproyecto SEEQ



Fuente: (Búrbano, Saravia, Sandoval, Suazo, & Morales, 2011)

La interfaz de datos RS-232 desarrollada en este trabajo de graduación permite integrar el radar laser Hokuyo URG-04LX-UG01 en aplicaciones de mapeo y exploración en tiempo real como esta sin necesidad de utilizar una computadora convencional como anfitrión USB para controlar el sensor.

XI. DISCUSIÓN

A través de la realización de esta tesis se implementó un sistema de reconocimiento de entorno para aplicaciones robóticas utilizando un radar láser de barrido como el Hokuyo URG-04LX-UG01. La topología del estándar USB establece que los dispositivos periféricos, como el URG-04LX-UG01, pueden comunicarse únicamente con un anfitrión USB. Este se encarga de enumerar a los periféricos conectados y de iniciar todas las transferencias de datos a través del bus de comunicación. Por esta razón, fue necesario implementar un anfitrión USB en el microcontrolador PIC24FJ64GB002 para poder comunicarse con el sensor láser a través de su interfaz de datos USB 2.0. Dicho anfitrión USB permite transmitir comandos de adquisición de datos hacia el sensor láser y procesar los datos de distancia contenidos en la respuesta del mismo.

El anfitrión USB implementado en el microcontrolador es del tipo "dedicado". Los anfitriones USB dedicados soportan una sola clase de dispositivos periféricos y solo algunos tipos de transferencias de datos. Por esta razón, pueden ser implementados en microarquitecturas con recursos limitados de procesamiento y memoria como la de un microcontrolador. El anfitrión USB dedicado implementado en el PIC24FJ64GB002 soporta la conexión únicamente de dispositivos periféricos del tipo *"Communications Device Class"* (CDC) e implementa únicamente las transferencias de datos de control y las masivas. Dicho anfitrión USB dedicado permite comunicarse satisfactoriamente con el sensor láser a través de su interfaz de datos USB 2.0.

Sin embargo, se determinó experimentalmente que, a diferencia de una computadora, los recursos de procesamiento y memoria del anfitrión USB dedicado no son suficientes para procesar satisfactoriamente los datos del sensor láser cuando este último es configurado a su máximo desempeño.

Es decir, cuando el sensor es configurado a su máximo alcance (5,600 mm), máxima resolución angular (0.35°) y máxima frecuencia de barrido (10 Hz) simultáneamente. Por esta razón, el anfitrión USB dedicado fue configurado para limitar el desempeño y la tasa de generación de datos del sensor láser con el objetivo de reducir al mínimo la cantidad de errores de procesamiento de los datos derivados de la limitante de procesamiento y memoria del microcontrolador.

Luego, se determinó que el desempeño y la tasa de generación de datos del sensor láser dependen de varios factores que incluyen: el número de bytes utilizados para la codificación de los datos de distancia, la resolución angular configurada y la frecuencia de transmisión de datos hacia el anfitrión USB dedicado.

Con respecto al número de bytes utilizados para la codificación de los datos, el protocolo de comunicación del sensor láser SCIP2.0 establece que los datos de distancia de hasta 12 bits de longitud son codificados utilizando 2 caracteres ASCII. La distancia más grande que puede ser codificada de esta forma es 2^12 - 1 = 4,095 mm. Entonces, cada uno de los datos de distancia es codificado en una cadena de 2 caracteres ASCII de la forma "XY". Este algoritmo de codificación es utilizado por el comando de adquisición de datos "MS" y limita el alcance del sensor al 73% (4,095 mm) del máximo.

Además, según el protocolo SCIP2.0, los datos de distancia de hasta 18 bits de longitud son codificados utilizando 3 caracteres ASCII. La distancia más grande que puede ser codificada de esta forma es 2^18 - 1 = 262,143 mm. Sin embargo, la distancia más grande que puede medir el sensor láser URG-04LX-UG01 es de 5,600 mm. Entonces, cada uno de los datos de distancia es codificado en una cadena de 3 caracteres ASCII de la forma "XYZ". Este algoritmo de codificación es utilizado por el comando de adquisición de datos "MD" y permite utilizar el 100% (5,600 mm) del alcance del sensor láser.

Con respecto a la resolución angular, se determinó que cuando esta se configura a su valor más alto, el sensor láser genera 682 (240°/0.3515°) datos de distancia en cada barrido de medición. Además, según la hoja de especificaciones del sensor láser, este puede efectuar un barrido de medición cada 100 ms. Entonces, cuando se transmite el comando de adquisición de datos *"MS"* al sensor, este codifica cada uno de los 682 datos de distancia utilizando 2 caracteres ASCII (2 bytes) y la tasa de generación de datos obtenida es aproximadamente 13 kB/s (682*2 Bytes / 100 ms).

Por otro lado, cuando se transmite el comando de adquisición de datos "MD" al sensor, este codifica cada uno de los 682 datos de distancia utilizando 3 caracteres ASCII (3 bytes) y la tasa de generación de datos en este caso es aproximadamente 20 kB/s (682*3 Bytes / 100 ms).

Luego, con respecto a la frecuencia de transmisión de los datos hacia el anfitrión USB, los comandos de adquisición de datos definidos en el protocolo de comunicación SCIP2.0 (MS, MD) poseen un parámetro de configuración llamado *"Scan Interval"* que permite configurar un número de barridos de medición omitidos periódicamente por el sensor láser (entre 1-9) y modificar así la frecuencia de transmisión de datos hacia el microcontrolador.

Tomando en cuenta lo anterior, se determinó experimentalmente que el anfitrión USB dedicado implementado en el PIC24FJ64GB002 es capaz de decodificar y procesar satisfactoriamente los datos de distancia contenidos en la respuesta del sensor láser mientras su tasa de generación de datos no exceda los 6.8 kB/s.

Para limitar el desempeño y la tasa de generación de datos del sensor láser de tal forma que esta no exceda el máximo permitido se decidió reducir el número de bytes utilizados para la codificación de los datos de distancia y la frecuencia de transmisión de datos hacia el microcontrolador.

Para ello, el anfitrión USB dedicado fue configurado para requerir los datos de distancia al sensor láser utilizando el comando de adquisición de datos "MS". Este comando, en comparación con el comando "MD", reduce aproximadamente 1kB la cantidad de datos generados por el sensor en cada barrido de medición. Por esta razón, la distancia más grande que puede ser codificada por el sensor láser cuando es controlado por al anfitrión USB dedicado en el PIC24FJ64GB002 es 4,095 mm.

Además, el parámetro *"Scan Interval"* del comando de adquisición de datos "MS" transmitido al sensor fue configurado con el valor "1". Esto provoca que el sensor láser omita periódicamente 1 de cada 2 barridos de medición consecutivos reduciendo así la frecuencia de transmisión de datos hacia el microcontrolador a la mitad de su valor original (10 Hz).

Las modificaciones realizadas provocan que el sensor codifique cada uno de los datos de distancia en un barrido de medición utilizando una cadena de 2 caracteres ASCII (2 bytes) y genere aproximadamente 1.3 kB de datos cada 200 ms (5 Hz). De esta forma la tasa de generación de datos del sensor láser fue limitada a (1.3/200) kB/ms ó 6.8 kB/s.

Por último, los algoritmos de clasificación, decodificación, detección y manejo de errores y procesamiento de los datos del sensor láser fueron implementados en el PIC24FJ64GB002 y no en la aplicación gráfica con el objetivo de crear una aplicación modular, independiente de una computadora, que permita incorporar el sensor láser de barrido URG-04LX-UG01 a la mayoría de aplicaciones robóticas.

XII. CONCLUSIONES

- Se implementó un anfitrión USB dedicado en el microcontrolador PIC24FJ64GB002 con soporte para dispositivos periféricos del tipo *"Communications Device Class"* (CDC) únicamente.

- El anfitrión USB implementado en el microcontrolador PIC24FJ64GB002 soporta únicamente las transferencias de datos de control y las masivas debido a las limitantes de procesamiento y memoria impuestas por el tipo de microarquitectura.

- La interfaz gráfica desarrollada permite visualizar los datos de distancia del sensor láser URG-04LX-UG01 en un plano de dos dimensiones y configurar gráficamente su resolución angular, rango mínimo y máximo y frecuencia de medición.

- El procesamiento y decodificación de los datos del sensor fueron implementados en el microcontrolador PIC24FJ64GB002 y no en la aplicación gráfica de tal forma que el radar láser puede ser integrado a la mayoría de aplicaciones robóticas sin necesidad de una computadora de tamaño regular que ejecute la aplicación gráfica y procese los datos.

- Se determinó experimentalmente que el anfitrión USB implementado en el PIC24FJ64GB002 es capaz de decodificar y procesar satisfactoriamente los datos de distancia del sensor láser con una tasa máxima de procesamiento de hasta 6.8 kB/s aproximadamente debido a sus limitantes de procesamiento.

- La interfaz de datos RS-232 desarrollada implementa el comando *"MS"* definido en el protocolo de comunicación del radar láser SCIP2.0 para la obtención de datos de distancia. Este comando permite realizar mediciones de distancia en un rango de detección de 240° con un alcance máximo de hasta 4.09 metros.

  - Se utilizó el comando de adquisición de datos "MS" y el parámetro "Scan Interval" para limitar la tasa de generación de datos de distancia del sensor a 6.8 kB/s.

XIII. RECOMENDACIONES

- Se recomienda modificar la resolución angular del sensor láser para determinar experimentalmente el valor adecuado para una aplicación en específico de tal forma que el contorno de medición sea continuo y no contenga variaciones abruptas generadas por una excesiva resolución angular.

- Se recomienda utilizar una fuente de alimentación regulada de 5 VDC que pueda suministrar por lo menos 1A de corriente al circuito impreso para garantizar el correcto funcionamiento del sensor láser y evitar dañar su circuitería interna.

- Se recomienda implementar la decodificación de 3 caracteres en el procesamiento de los datos del sensor láser para aumentar su alcance máximo hasta 5.60 metros.

- Se recomienda implementar la interfaz de datos RS-232 para el radar láser Hokuyo URG-04LX-UG01 en un ordenador de placa reducida o *Single Board Computer* (SBC) que, a diferencia de los microcontroladores, posee una microarquitectura basada en un microprocesador con recursos de procesamiento y memoria suficientes para decodificar los datos de distancia del sensor con una tasa de procesamiento de hasta 20.5 kB/s aproximadamente.

XIV. BIBLIOGRAFÍA

Acroname Robotics. (4 de Marzo de 2011). *Laser Range Finder Overview.* Obtenido de sitio web de Acroname: http://www.acroname.com/robotics/info/articles/laser/laser.html#c

Búrbano, M. (Febrero de 2012). *Wireless RS-232 Data Interface for Simple-URG.*

Búrbano, M., Saravia, R., Sandoval, M., Suazo, D., & Morales, J. (2011). *Megaproyecto SEEQ Robohelicóptero.* Guatemala.

Carletti, E. J. (2010). *Sensores-Reflectivos y por intercepción.* Obtenido de Robots Argentina: http://robots-argentina.com.ar/Sensores_reflectivos.htm

Colombo, G., & Cecic, D. (2011). *COM 3101: Introduction to Full-Speed USB.* Obtenido de Microchip Regional Training Centers.

Esquit, C. (2010). *IE 3003: Instrumentación electrónica: Lecture 2.* Obtenido de Caracterización de instrumentos - Representación en diagramas de flujo.

Gupta, A. (4 de Febrero de 2009). *AN1247: Communication Device Class (CDC) Host.* Obtenido de Microchip Technology Inc.

Hokuyo. (s.f.). *URG-04LX-UG01 (Simple URG): Technical Drawing.* Obtenido de sitio web de Hokuyo: URG Series Download Page: http://www.hokuyo-aut.jp/02sensor/07scanner/download/data/URG-04LX_UG01_ed.pdf

Hokuyo. (10 de Octubre de 2006). *Communication Protocol Specification for SCIP2.0 Standard.* Obtenido de sitio web de Hokuyo: http://www.hokuyo-aut.jp/02sensor/07scanner/download/data/URG_SCIP20.pdf

Hokuyo. (20 de Agosto de 2009). *Scanning Laser Range Finder URG-04LX-UG01: Specifications.* Obtenido de sitio web de Hokuyo: http://www.hokuyo-aut.jp/02sensor/07scanner/download/data/URG-04LX_UG01_spec.pdf

Hokuyo. (2009). *Scanning Range Finder: URG-04LX-UG01.* Obtenido de sitio web de Hokuyo: http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html

Hokuyo. (21 de Febrero de 2012). *URG programming guide: Explanation of SCIP commands .* Obtenido de sitio web de Hokuyo: http://www.hokuyo-aut.jp/02sensor/07scanner/download/urg_programs_en/scip_commands_page.html

Lasermet, Ltd. (2012). *An Overview of the LED and Laser Classification System in EN 60825-1 and IEC 60825-1.* Obtenido de sitio web de Lasermet: http://www.lasermet.com/resources/classification_overview.php

Metrology Resource Co. (2010). *Metrology Resource Products - Distance Laser Sensors.* Obtenido de sitio web de Metrology Resource Co.: http://www.metrologyresource.com/laser-sensor-MRL3.php

Microchip. (Enero de 2010). *PIC24F Family Reference Manual Section 27: USB On-The-Go (OTG).* Obtenido de Microchip Technology Inc.

Microchip. (2010). *PIC24FJ64GB004 Familiy Data Sheet.* Obtenido de Microchip.
Microchip Technology. (2008). *USB Overview.* Obtenido de Microchip Design Center - USB Introduction: http://ww1.microchip.com/downloads/en/DeviceDoc/General%20USB%20ver%201.1.pdf

Microsoft. (2005). *Managing Devices.* Obtenido de USB Tophology: http://technet.microsoft.com/en-us/library/bb457107.aspx

Otten, D., Cowden, S., & Budagutta, P. (8 de Febrero de 2011). *AN1095: Emulating Data EEPROM for PIC18 and PIC24 Microcontrollers and dsPIC Digital Signal Controllers.* Obtenido de Microchip Technology Inc.

Otten, K. (2 de Enero de 2008). *AN1140: USB Embedded Host Stack.* Obtenido de Microchip Technology Inc.

Parallax. (Junio de 2005). *Sharp GP2D12 Analog Distance Sensor.* Obtenido de sitio web de Parallax: http://www.parallax.com/dl/docs/prod/acc/SharpGP2D12Snrs.pdf

Peacock, C. (17 de Septiembre de 2010). *USB in a NutShell: Making sense of the USB standard.* Obtenido de sitio de Beyond Logic: http://www.beyondlogic.org/usbnutshell/usb1.shtml

PINOUTS.RU. (21 de Julio de 2011). *USB pinout.* Obtenido de sitio web de PINOUTS.RU: http://pinouts.ru/Slots/USB_pinout.shtml

Rockwell Laser Industries. (2001). *Laser Standards and Classifications: IEC.* Obtenido de sitio web de RLI: http://www.rli.com/resources/articles/classification.aspx

SICK. (2012). *Laser measurement technology.* Obtenido de LMS 100: https://www.mysick.com/ecat.aspx?go=FinderSearch&Cat=Row&At=Fa&Cult=English&FamilyID=344&Category=Produktfinder&Selections=34242

Society of Robots. (2005). *Sensors - Sharp IR range finder.* Obtenido de sitio web de Society of Robots: http://www.societyofrobots.com/sensors_sharpirrange.shtml

Universidad de Princeton. (3 de Octubre de 2007). *Section 3: Laser control measures.* Obtenido de Environmental Health and Safety: http://web.princeton.edu/sites/ehs/laserguide/sec3.htm#top

USB made simple. (2010). *Part 1 - Introduction to USB.* Obtenido de General Introduction: http://www.usbmadesimple.co.uk/ums_1.htm

USB.org. (s.f.). *Introduction to USB On-The-Go.* Obtenido de sitio web de USB.org: http://www.usb.org/developers/onthego/USB_OTG_Intro.pdf

USB.org. (18 de Diciembre de 2001). *On-The-Go Supplement to the USB 2.0 Specification.* Obtenido de sitio de USB.org: http://www.usb.org/developers/onthego/otg1_0.pdf

Wikipedia. (1 de Abril de 2012). *Laser safety.* Obtenido de sitio web de Wikipedia: http://en.wikipedia.org/wiki/Laser_safety

Wikipedia. (9 de Abril de 2012). *Universal Serial Bus.* Obtenido de USB: http://en.wikipedia.org/wiki/Universal_Serial_Bus

Wright State University. (2010). *Intelligent Sensor Systems: .* Obtenido de Lecture 2: Sensor characteristics: http://research.cs.tamu.edu/prism/lectures/iss/iss_l2.pdf

XV. APÉNDICE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
using System.Collections;
using System.Drawing.Printing;
using System.IO;
using System.IO.Ports;
using System.Runtime.InteropServices;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        private string msj = "", templine = "";
        private int[] distancias = new int [682];
        private int[] angulos = new int[682];
        private int contDist = 0, msegs = 0, numdatos = 682;
        private double angRES = 0.0;

        private byte[] RxData = new byte[5];
        private int tempD = 0, clusterCount = 1, startStep, endStep, grados, angleStart, angleSweep;
        private const int STARTING_STEP = 44;

        Graphics radar, preview;
        Bitmap bmap, bmap2;

        int width, height, width2, height2;

        public Form1()
        {
            InitializeComponent();

            for (int i = 0; i < distancias.Length; i++)
            {
                distancias[i] = 0;
            }

            button2.Text = "Conectar (" + HOKUYO.PortName + ")";

            radar = pictureBox1.CreateGraphics();
            width = pictureBox1.Width;
            height = pictureBox1.Height;
            dibuja_radar();

            preview = pictureBox2.CreateGraphics();
            width2 = pictureBox2.Width;
            height2 = pictureBox2.Height;
            dibuja_preview();
        }

        private void dibuja_radar()
        {
            bmap = new Bitmap(width, height, radar);
            radar = Graphics.FromImage(bmap);

            Pen myPen = new Pen(Color.Gray);
            myPen.DashStyle = DashStyle.Dot;

            radar.DrawLine(myPen, width / 2, 0, width / 2, height);
            radar.DrawLine(myPen, 0, height / 2, width, height / 2);
            radar.DrawLine(myPen, width / 2, height / 2, 0, height);
            radar.DrawLine(myPen, width / 2, height / 2, width, height);
```

```
        myPen = new Pen(Color.Blue);
        myPen.Width = 1.0F;

        angRES = (360.0/1024.0)*clusterCount;

        for (int i = 0; i < numdatos-1; i++)
        {
            int x1 = (int)(width / 2 + 0.5 * distancias[i] / 10.0 * Math.Cos((getGrados() + angRES * i) * ↙
 Math.PI / 180.0));
            int y1 = (int)(height / 2 - 0.5 * distancias[i] / 10.0 * Math.Sin((getGrados() + angRES * i)  ↙
* Math.PI / 180.0));

            int x2 = (int)(width / 2 + 0.5 * distancias[i + 1] / 10.0 * Math.Cos((getGrados() + angRES *  ↙
(i + 1)) * Math.PI / 180.0));
            int y2 = (int)(height / 2 - 0.5 * distancias[i + 1] / 10.0 * Math.Sin((getGrados() + angRES * ↙
 (i + 1)) * Math.PI / 180.0));

            radar.DrawLine(myPen, x1, y1, x2, y2);
        }
        pictureBox1.Image = bmap;
    }

    private int getGrados()
    {
        grados = (int)((startStep - STARTING_STEP)/clusterCount * angRES - 30);

        return grados;
    }

    private void dibuja_preview()
    {
        bmap2 = new Bitmap(width2, height2, preview);
        preview = Graphics.FromImage(bmap2);

        Pen myPen2 = new Pen(Color.Gray);
        myPen2.DashStyle = DashStyle.Dot;

        preview.DrawLine(myPen2, width2 / 2, 0, width2 / 2, height2);
        preview.DrawLine(myPen2, 0, height2 / 2, width2, height2 / 2);
        preview.DrawLine(myPen2, width2 / 2, height2 / 2, 0, height2);
        preview.DrawLine(myPen2, width2 / 2, height2 / 2, width2, height2);

        if (textBox2.Text != "" && textBox3.Text != "")
        {
            startStep = Convert.ToInt16(textBox2.Text, 10);
            endStep = Convert.ToInt16(textBox3.Text, 10);
            angleStart = (int)(((startStep - STARTING_STEP) * (360.0 / 1024) - 30) * -1);
            angleSweep = (int)(((endStep - startStep) * (360.0 / 1024)) * -1);

            SolidBrush myBrush = new SolidBrush(Color.Blue);
            preview.FillPie(myBrush, width2 / 4, height2 / 4, width2 / 2, width2 / 2, angleStart,        ↙
angleSweep);
        }
        pictureBox2.Image = bmap2;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        msj = textBox1.Text + textBox2.Text + textBox3.Text + textBox4.Text + textBox5.Text + textBox6.   ↙
Text;
        label7.Text = msj;

        if(HOKUYO.IsOpen)
            HOKUYO.WriteLine(msj);

        startStep = Convert.ToInt16(textBox2.Text, 10);
        clusterCount = Convert.ToInt16(textBox4.Text, 10);
```

```
        }

        private void button6_Click(object sender, EventArgs e)
        {
            if (HOKUYO.IsOpen)
                HOKUYO.WriteLine(textBox7.Text);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            if (HOKUYO.IsOpen == false)
            {
                HOKUYO.Open();
                button2.Text = "Desconectar";
            }
            else
            {
                HOKUYO.Close();
                button2.Text = "Conectar (" + HOKUYO.PortName + ")";
            }
        }

        private void HOKUYO_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            this.Invoke(new EventHandler(DoUpdate));
        }

        private void DoUpdate(object s, EventArgs e)
        {
            do
            {
                if (tabControl1.SelectedTab == tabPage1)
                {
                    if (HOKUYO.BytesToRead >= 2)
                    {
                        HOKUYO.Read(RxData, 0, 2);
                        analizarVIEWER();
                    }
                }
                else
                    if (tabControl1.SelectedTab == tabPage2 || tabControl1.SelectedTab == tabPage3)
                    {
                        templine = HOKUYO.ReadLine();
                        analizarCONFIG();
                    }
            }
            while (HOKUYO.BytesToRead > 0);
        }

        private void analizarCONFIG() {

            if (tabControl1.SelectedTab == tabPage2) {

                richTextBox1.AppendText(templine + "\n");

                if (templine.Length >= 10 && templine.Substring(0, 10) == "Numdatos: ")
                {
                    numdatos = Convert.ToInt16(templine.Substring(10, 4), 10);
                    richTextBox1.AppendText("Numdatos recibido: " + numdatos + "\n\n");
                }
            }

            if (tabControl1.SelectedTab == tabPage3) {
                richTextBox4.AppendText(templine + "\n");
                //richTextBox4.AppendText(templine + " (" + (templine.Length -1) + ")\n");
            }
        }
```

```
        private void analizarVIEWER() {

            tempD = RxData[0];
            tempD += (RxData[1] << 8);

            distancias[contDist] = tempD;
            contDist++;

            if (contDist >= numdatos)
                contDist = 0;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            richTextBox4.Clear();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            dibuja_radar();
        }

        private void button5_Click(object sender, EventArgs e)
        {
            tabControl1.SelectedTab = tabPage1;

            contDist = 0;
            for (int i = 0; i < distancias.Length; i++)
                distancias[i] = 0;

            if (HOKUYO.IsOpen)
                HOKUYO.WriteLine("GO");
            timer1.Enabled = true;
        }

        private void button4_Click(object sender, EventArgs e)
        {
            richTextBox1.Clear();
        }

        private void textBox2_TextChanged(object sender, EventArgs e)
        {
            dibuja_preview();
        }

        private void textBox3_TextChanged(object sender, EventArgs e)
        {
            dibuja_preview();
        }
    }
}
```

```
/*
Universidad del Valle de Guatemala
Autor: Mario Andres Burbano Castro
Carne: 07155

Fecha: 23-feb-2012
Descripcion: Driver CDC para sensor laser Hokuyo
Version: 20
Procesador: PIC24FJ64GB002
*/

#include <p24FJ64GB002.h>
#include <libpic30.h>
#include <uart.h>
#include <PPS.h>
#include <timer.h>

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "GenericTypeDefs.h"
#include "HardwareProfile.h"
#include "usb_config.h"
#include "USB/usb.h"
#include "USB/usb_host_cdc.h"
#include "USB/usb_host_cdc_interface.h"

//DEE Emulation 16-bit
//#include <.\DEE Emulation 16-bit\DEE Emulation 16-bit.h>
#include <DEE Emulation 16-bit.h>

/*-----------------------------------------------------------------------
-------------------------/
/                                    PALABRAS DE CONFIGURACION
                                        /
/-----------------------------------------------------------------------
-------------------------*/

_CONFIG1(WDTPS_PS1 & FWPSA_PR32 & WINDIS_OFF & FWDTEN_OFF & ICS_PGx1 &
GWRP_ON & GCP_OFF & JTAGEN_OFF)
_CONFIG2(POSCMOD_EC & I2C1SEL_PRI & IOL1WAY_OFF & OSCIOFNC_OFF &
FCKSM_CSDCMD & FNOSC_PRIPLL & PLL96MHZ_ON & PLLDIV_DIV5 & IESO_OFF)
_CONFIG3(WPFP_WPFP0 & SOSCSEL_IO & WUTSEL_LEG & WPDIS_WPDIS &
WPCFG_WPCFGDIS & WPEND_WPENDMEM)
_CONFIG4(DSWDTPS_DSWDTPS0 & DSWDTOSC_LPRC & RTCOSC_SOSC & DSBOREN_OFF &
DSWDTEN_OFF)


/*-----------------------------------------------------------------------
-------------------------/
/                                          DATA STRUCTURES
                                              /
```

```
    /------------------------------------------------------------------------
    -------------------------*/

    typedef enum _APPL_STATE{

        DEMO_INITIALIZE,
        DEVICE_NOT_CONNECTED,
        DEVICE_CONNECTED,           /* Device Enumerated  - Report Descriptor
    Parsed */
        READY_TO_TX_RX,
        GET_IN_DATA,
        GET_IN_DATA_WAIT,
        SEND_OUT_DATA,
        SEND_OUT_DATA_WAIT,
        ERROR_REPORTED,             /* need to add application states for data
    interface */
        CONFIG_SEND_OUT_PC,
        SEND_OUT_PC_WAIT

    } APPL_STATE;

    typedef enum _SENSOR_MODE{

      CONFIG,
      TEST,
      RUNNING

    } SENSOR_MODE;


    /*------------------------------------------------------------------------
    -------------------------/
    /                                              MACROS
                                                       /
    /------------------------------------------------------------------------
    -------------------------*/
    #define MAX_ALLOWED_CURRENT             (500)         // Maximum power we
    can supply in mA
    #define MAX_NO_OF_IN_BYTES  64
    #define MAX_NO_OF_OUT_BYTES 64


    /*------------------------------------------------------------------------
    -------------------------
                                           DECLARACION DE VARIABLES
    ------------------------------------------------------------------------
    -------------------------*/

    volatile APPL_STATE  APPL_Demo_State;
    volatile SENSOR_MODE RADAR_Mode = TEST;

    BYTE USB_CDC_IN_Data_Array[MAX_NO_OF_IN_BYTES];
```

```
    BYTE USB_CDC_OUT_Data_Array[MAX_NO_OF_OUT_BYTES];
    BYTE ErrorDriver;
    BYTE NumOfBytesRcvd;
    BYTE contline = 0, contdatos = 0, count = 0, w = 0;

    BOOL ENVIARaHOKUYO = FALSE;
    BOOL ENVIANDOaPC = FALSE;
    BOOL displayOnce = FALSE;
    BOOL displayOnce1 = FALSE;
    BOOL LF1 = FALSE;
    BOOL TxError = FALSE;

    char endcmd1[] = "ENDCMD\n";
    char newline[] = "\n";
    char sumOK[] = "SUM OK\n";
    char sumERR[] = "ERROR en SUM\n";
    char prueba[] = "Hokuyoo";
    char sum[] = "#\n";
    char MScmd[] = "MS0044072500101\n";
    char templine[70];
    char Rxdata[17];
    //char dato[] = "xxxx\n";
    char dato[] = "#\n#\n#\n";

    unsigned int contDistancias = 0, segs_1 = 0, segs_15 = 0;
    unsigned int startStep = 44, endStep = 725, clusterCount = 1, scanInterval
    = 1, numberOfScans = 1;
    unsigned int numdatos = 682, residuo = 0;
    unsigned int distancias[682];

    //DEE Emulation 16-bit
    unsigned int value1, value2, value3;
    unsigned int DEEdata = 0;
    unsigned int DEEaddr1 = 4, DEEaddr2 = 261, DEEaddr3 = 302;
    BYTE error1;


    /*-------------------------------------------------------------------------
    -------------------------/
    /                                  USB APPLICATION EVENT HANDLER
                                          /
    /-------------------------------------------------------------------------
    -------------------------*/

    BOOL USB_ApplicationEventHandler( BYTE address, USB_EVENT event, void
    *data, DWORD size )
    {
        switch( event )
        {
            case EVENT_VBUS_REQUEST_POWER:
                // The data pointer points to a byte that represents the
    amount of power
```

```
                // requested in mA, divided by two.  If the device wants too
    much power,
                // we reject it.
                if (((USB_VBUS_POWER_EVENT_DATA*)data)->current <=
    (MAX_ALLOWED_CURRENT / 2))
                {
                    return TRUE;
                }
                else
                {
                    //***** USB Error - device requires too much current *****
                    putsUART1((unsigned int *)"USB Error - Device requires too
    mucho current\n");
                }
                break;

            case EVENT_VBUS_RELEASE_POWER:
                // Turn off Vbus power.
                // The PIC24F with the Explorer 16 cannot turn off Vbus
    through software.
                putsUART1((unsigned int *)"USB Event - Power released on
    Vbus\n");
                return TRUE;
                break;

            case EVENT_HUB_ATTACH:
                //***** USB Error - hubs are not supported *****
                putsUART1((unsigned int *)"USB Error - Hubs are not
    supported\n");
                return TRUE;
                break;

            case EVENT_UNSUPPORTED_DEVICE:
                //***** USB Error - device is not supported *****
                putsUART1((unsigned int *)"USB Error - Device is not
    supported\n");
                return TRUE;
                break;

            case EVENT_CANNOT_ENUMERATE:
                //***** USB Error - cannot enumerate device *****
                putsUART1((unsigned int *)"USB Error - Cannot enumerate
    device\n");
                return TRUE;
                break;

            case EVENT_CLIENT_INIT_ERROR:
                //***** USB Error - client driver initialization error *****
                putsUART1((unsigned int *)"USB Error - Client driver
    initialization error\n");
                return TRUE;
                break;
```

```
            case EVENT_OUT_OF_MEMORY:
                //***** USB Error - out of heap memory *****
                putsUART1((unsigned int *)"USB Error - Out of heap memory\n");
                return TRUE;
                break;

            case EVENT_UNSPECIFIED_ERROR:   // This should never be generated.
                //***** USB Error - unspecified *****
                putsUART1((unsigned int *)"USB Error - Unspecified\n");
                return TRUE;
                break;

            case EVENT_DETACH:                  // USB cable has been
    detached (data: BYTE, address of device)
                putsUART1((unsigned int *)"USB Event - USB cable has been
    detached\n");
                displayOnce = FALSE;
                displayOnce1 = FALSE;
                return TRUE;
                break;

    // CDC Specific events
            case EVENT_CDC_NONE:
            case EVENT_CDC_ATTACH:
            case EVENT_CDC_COMM_READ_DONE:
            case EVENT_CDC_COMM_WRITE_DONE:
            case EVENT_CDC_DATA_READ_DONE:
            case EVENT_CDC_DATA_WRITE_DONE:
            case EVENT_CDC_NAK_TIMEOUT:
            case EVENT_CDC_RESET:

            default :
                break;
        }
        return FALSE;
    }

    void USBHostCDC_Clear_Out_DATA_Array(void)
    {
        BYTE i;

        for(i=0;i<MAX_NO_OF_OUT_BYTES;i++)
            USB_CDC_OUT_Data_Array[i] = 0;
    }

    void USBHostCDC_Clear_In_DATA_Array(void)
    {
        BYTE j;

        for(j=0;j<MAX_NO_OF_IN_BYTES;j++)
            USB_CDC_IN_Data_Array[j] = 37;
```

```
    }

    void Clear_Rxdata_Array(void)
    {
        BYTE k;

        for(k=0;k<sizeof(Rxdata);k++)
            Rxdata[k] = 0;
    }

    void Clear_templine(void)
    {
        BYTE m;

        for(m=0; m<sizeof(templine); m++)
            templine[m] = 0;
    }

    void Clear_distancias(void){
        int n;

        for(n=0; n<sizeof(distancias)/2; n++)
          distancias[n] = 0;
    }

    /*-----------------------------------------------------------------------
    -------------------------
                                            PROCEDIMIENTOS
    -----------------------------------------------------------------------
    -------------------------
    Definicion del procedimiento            Funcion

    - void togglePort(void)                 Hace 'toggle' del estado del
    puerto A; PORTA<4:0>
    - void configUART1(void)                Configura e inicia el modulo
    periferico UART1
    - void configIOPORTS(void)              Configura el estado de los puertos
    de entrada/salida
    -----------------------------------------------------------------------
    -------------------------*/

    void configTIMER1(void)
    {
       unsigned int CONFIGvalue;
       unsigned int PR1value = 10;

       CONFIGvalue =  T1_ON  &
                      T1_IDLE_STOP &
                      T1_GATE_OFF  &
                      T1_PS_1_64  &
                      T1_SYNC_EXT_OFF &
                      T1_SOURCE_INT;
```

```
   ConfigIntTimer1(T1_INT_PRIOR_1 & T1_INT_ON);         //Enable
Interrupt*/
   OpenTimer1(CONFIGvalue, PR1value);                   //Timer is
configured for 10 msec
}

void configUART1(void)
{
    unsigned int baudvalue = 8;
    unsigned int U1MODEvalue;
    unsigned int U1STAvalue;

    /*-------------------------------------------------------------------
                        PARAMETROS CONFIG. UART1
    -------------------------------------------------------------------
            - 115200 baudrate
            - Low baud rate
            - 8 bit transmission/reception
            - No parity bit
            - 1 stop bit
            - U1RX pin: RB0
            - U1TX pin: RB1
    -------------------------------------------------------------------*/

    iPPSInput(IN_FN_PPS_U1RX,IN_PIN_PPS_RP3);          //Assing U1RX to pin

    iPPSOutput(OUT_PIN_PPS_RP2,OUT_FN_PPS_U1TX );      //Assing U1TX to pin



    CloseUART1();                                      // Reset the UART
module

    // UART Interrupt setup-----------------------------------------------

    ConfigIntUART1( UART_RX_INT_EN &                   // UART Receive
Interrupts
                    UART_RX_INT_PR7 &                  // Assign Priority
7 to RX interrupt
                    UART_TX_INT_EN &                   // UART Transmit
Interrupts
                    UART_TX_INT_PR6);                  // Assign Priority
6 to TX interrupts

    // UxMODE Register----------------------------------------------------
    U1MODEvalue = UART_EN &                            // Enable the UART
module
                  UART_IDLE_CON &                      // UART continues
operation in debug mode
                  UART_IrDA_DISABLE &                  // Disable IRDA
mode
```

```
                    UART_MODE_SIMPLEX &                    // Disable RTS/CTS
handshaking
                    UART_UEN_00 &                          // tx / rx only. No
cts or rts
                    UART_DIS_WAKE &                        // Disable sleep
mode wake up on start bit reception
                    UART_DIS_LOOPBACK &                    // Disable loopback
mode
                    UART_DIS_ABAUD &                       // Disable auto
baud rate measurement
                    UART_UXRX_IDLE_ONE &                   // UxRx Idle state
is one!!
                    UART_BRGH_SIXTEEN &                    // Use low speed
BRG (high speed one is silicon bugged)
                    UART_NO_PAR_8BIT &                     // 8-bit data, no
parity
                    UART_1STOPBIT;                         // 1-stop bit

    // UxSTA Register-----------------------------------------------------
    U1STAvalue = UART_INT_TX_BUF_EMPTY  &                  // Interrupt when
shift-register and buffer are empty
                    UART_IrDA_POL_INV_ZERO &               // TX idle state is
zero (no inversion)
                    UART_SYNC_BREAK_DISABLED &             // Disable transmit
break transmission
                    UART_TX_ENABLE &                       // Enable the UART
Transmission block
                    UART_INT_RX_CHAR  &                    // RX Interrupt on
every character received
                    UART_ADR_DETECT_DIS &                  // Disable address
detect feature
                    UART_RX_OVERRUN_CLEAR;                 // Clear any overun
error state


    // UART Enable/Open---------------------------------------------------
    OpenUART1(U1MODEvalue, U1STAvalue, baudvalue);
}

void configIOPORTS(void)
{
    AD1PCFG = 0b000111111111111;
    TRISA = 0;
    PORTA = 0;
}

void configCMD(void){

  unsigned int i;                  // Starting step (4 bytes) - MScmd<2:5>
  for(i=2; i<15; i++)              // End step (4 bytes) - MScmd<6:9>
    MScmd[i] = Rxdata[i];          // Cluster count (2 bytes) -
MScmd<10:11>
```

```
  MScmd[12] = '1';                  // Scan interval (1 byte) - MScmd<12>
                                    // Number of scans (2 bytes) -
MScmd<13:14>

  startStep = (MScmd[2]-0x30)*1000 + (MScmd[3]-0x30)*100 +
(MScmd[4]-0x30)*10 + (MScmd[5]-0x30);
  endStep = (MScmd[6]-0x30)*1000 + (MScmd[7]-0x30)*100 +
(MScmd[8]-0x30)*10 + (MScmd[9]-0x30);
  clusterCount = (MScmd[10]-0x30)*10 + (MScmd[11]-0x30);
  scanInterval = (MScmd[12]-0x30);
  numberOfScans = (MScmd[13]-0x30)*10 + (MScmd[14]-0x30);

  numdatos = ((endStep - startStep)+1)/clusterCount;
}

void notificar(void){
  /*
  putsUART1((unsigned int *)"Host CDC Demo: CONFIG updated\n");
  putsUART1((unsigned int *)"Numdatos: ");

  dato[0] = (numdatos/1000)+ 0x30;
  residuo = numdatos%1000;
  dato[1] = (residuo/100)+ 0x30;
  residuo = residuo%100;
  dato[2] = (residuo/10)+ 0x30;
  residuo = residuo%10;
  dato[3] = residuo+ 0x30;

  putsUART1((unsigned int *)dato);*/
}

unsigned int decodechar(char c1, char c2){

  int tempDist;
  BYTE* p = &tempDist;
  c1 -= 0x30;
  c2 -= 0x30;

  if((c1&1) == 1)
    c2|=(1<<6);
  else
    c2&=~(1<<6);
  if(((c1>>1)&1) == 1)
    c2|=(1<<7);
  else
    c2&=~(1<<7);
  c1 >>= 2;

  *p = c2;
  *(p+1) = c1;

  return tempDist;
```

```
    }

    void enviarDistancias(){
      unsigned int i;
      for(i = 0; i < numdatos; i++){
        WriteUART1(distancias[i]);
        WriteUART1(distancias[i]>>8);
        while(BusyUART1());
      }
    }

    void getDistancias(char a1[], int length1){
      int c;
      for(c= 0; c<length1/2; c++){
        distancias[contDistancias] = decodechar(a1[2*c], a1[2*c+1]);
        contDistancias++;
      }

      if(contDistancias >= numdatos){
        enviarDistancias();
        contDistancias = 0;
        Clear_distancias();
      }
    }

    BOOL endCMDf(){
      if(LF1 && USB_CDC_IN_Data_Array[w] != 10)
        LF1 = FALSE;
      if(!LF1 && USB_CDC_IN_Data_Array[w] == 10)
        LF1 = TRUE;
      else
        if(LF1 && USB_CDC_IN_Data_Array[w] == 10)
          return TRUE;

      return FALSE;
    }

    BOOL checkSUM(char a[], int length){
      unsigned int tempsum = 0, sum1 = 0;
      char SUM = a[length-1];

      int p;
      for(p=0; p<length-1; p++){
        tempsum += a[p];
      }
      sum1 = tempsum & 0b0000000000111111;
      sum1 += 0x30;

      if(SUM == sum1){
        return TRUE;
      }
      else{
```

```
        return FALSE;
      }
    }

    void procesar(void){

      if(checkSUM(templine, contdatos)){
        getDistancias(templine, contdatos-1);
      }
      else{
        TxError = TRUE;
        //WriteUART1('%');
      }
    }

    void analizar(void){

      for(w=0; w<MAX_NO_OF_IN_BYTES; w++){

        if(endCMDf()){

          if(RADAR_Mode == TEST)
            putsUART1((unsigned int *)newline);

          w = MAX_NO_OF_IN_BYTES;
          contdatos = 0;
          contline = 0;
          contDistancias = 0;
          Clear_distancias();
          if(TxError)
            TxError = FALSE;
        }
        else{
          if(USB_CDC_IN_Data_Array[w] != 10){
            templine[contdatos] = USB_CDC_IN_Data_Array[w];
            contdatos++;
          }

          if(USB_CDC_IN_Data_Array[w] == 10){

            if(RADAR_Mode == TEST){
              putsUART1((unsigned int *)templine);
              putsUART1((unsigned int *)newline);
            }
            contline++;

            if(contline>3 && RADAR_Mode == RUNNING && TxError == FALSE)
              procesar();

            contdatos = 0;
            Clear_templine();
          }
```

```
        }
      }
      USBHostCDC_Clear_In_DATA_Array();
    }

    void dataEEPROM(void){

        DEEdata = 37;
        error1 = DataEEWrite(DEEdata,0x4);
        dato[2] = error1 + 0x30;
        value1 = DataEERead(0x4);
        Nop();
        Nop();
        dato[4] = value1;
        putsUART1((unsigned int *)dato);
    }

    /*-------------------------------------------------------------------------
    -------------------------
                                    INTERRUPT SERVICE ROUTINES (ISR)
    -------------------------------------------------------------------------
    -------------------------*/

    void __attribute__ ((interrupt,no_auto_psv)) _U1RXInterrupt(void)
    {
        U1RX_Clear_Intr_Status_Bit;
        while(DataRdyUART1()){
          Rxdata[count++] = ReadUART1();
        }

        if((count == 16 && Rxdata[0] == 'M') || (count == 3 && Rxdata[0] ==
    'Q')){

          RADAR_Mode = TEST;
          ENVIARaHOKUYO = TRUE;
          PORTAbits.RA0 = ~PORTAbits.RA0;
          count = 0;
        }

        if(count == 16 && Rxdata[0] == '%'){

          RADAR_Mode = CONFIG;
          PORTAbits.RA0 = ~PORTAbits.RA0;
          count = 0;
          APPL_Demo_State = CONFIG_SEND_OUT_PC;
        }

        if(count == 3 && Rxdata[0] == 'G' && Rxdata[1] == 'O'){

          RADAR_Mode = RUNNING;
          ENVIARaHOKUYO = TRUE;
          PORTAbits.RA0 = ~PORTAbits.RA0;
```

```
            count = 0;
        }

        /*
        if(DataRdyUART1())
        {
            getsUART1(16,(unsigned int *)Rxdata, 1000);
            ENVIARaHOKUYO = TRUE;
            PORTAbits.RA0 = ~PORTAbits.RA0;

        }*/
    }

    void __attribute__ ((interrupt,no_auto_psv)) _U1TXInterrupt(void)
    {
        U1TX_Clear_Intr_Status_Bit;
        ENVIANDOaPC = FALSE;
    }

    void __attribute__ ((interrupt,no_auto_psv)) _T1Interrupt (void)
    {
        if(IFS0bits.T1IF){
          T1_Clear_Intr_Status_Bit;
          if(APPL_Demo_State == READY_TO_TX_RX)
          {
            APPL_Demo_State = GET_IN_DATA; // If no report is pending schedule
    new IN request
          }

          /*segs_1++;
          if(segs_1 == 25000){
            segs_1 = 0;
            segs_15++;

            if(segs_15 == 15){
              segs_15 = 0;
              PORTAbits.RA0 = 1;
              RADAR_Mode = RUNNING;
            }
          }*/
        }
    }

    /*-------------------------------------------------------------------------
    -------------------------
                                                MAIN
    -------------------------------------------------------------------------
    -------------------------*/

    int main(void)
    {
        configIOPORTS();
```

```
        configUART1();
        configTIMER1();

        ENVIARaHOKUYO = FALSE;
        ENVIANDOaPC = FALSE;
        displayOnce = FALSE;
        displayOnce1 = FALSE;

        USBHostCDC_Clear_In_DATA_Array();
        USBHostCDC_Clear_Out_DATA_Array();
        Clear_Rxdata_Array();
        Clear_templine();

        //DEE Emulation 16-bit
        error1 = DataEEInit();
        dato[0] = error1 + 0x30;
        dataEEFlags.val = 0;

        //Initialize USB layers
        USBInitialize( 0 );

        // Ciclo infinito
        while(1)
        {
            USBTasks();

            if(!USBHostCDC_ApiDeviceDetect()) // TRUE if device is enumerated
    without error
            {
               APPL_Demo_State = DEVICE_NOT_CONNECTED;
            }

            switch(APPL_Demo_State)
            {
                case DEMO_INITIALIZE:
                    APPL_Demo_State = DEVICE_NOT_CONNECTED;
                break;

                case DEVICE_NOT_CONNECTED:
                    USBTasks();
                    if(displayOnce == FALSE)
                    {
                       putsUART1((unsigned int *)"Host CDC Demo: Device Not
    connected\n");
                       displayOnce = TRUE;

                    }
                    if(USBHostCDC_ApiDeviceDetect()) // TRUE if device is
    enumerated without error
                    {
                       APPL_Demo_State = DEVICE_CONNECTED;
                    }
```

```
                break;

                case DEVICE_CONNECTED:
                    APPL_Demo_State = READY_TO_TX_RX;
                    putsUART1((unsigned int *)"Host CDC Demo: Device
    connected\n");
                break;

                case READY_TO_TX_RX:
                    if(ENVIARaHOKUYO == TRUE)
                    {
                         APPL_Demo_State = SEND_OUT_DATA;


                         BYTE i;

                         if(RADAR_Mode == TEST){
                           for( i=0; i < sizeof(Rxdata);i++)
                             USB_CDC_OUT_Data_Array[i] = Rxdata[i]; // Copy
    data to be transmitted into output buffer

                         }

                         if(RADAR_Mode == RUNNING){
                           for( i=0; i < sizeof(MScmd);i++)
                             USB_CDC_OUT_Data_Array[i] = MScmd[i]; // Copy
    data to be transmitted into output buffer

                         }
                    }

                    if(displayOnce1 == FALSE)
                    {
                        putsUART1((unsigned int *)"Host CDC Demo:
    READY_TO_TX_RX\n");
                        displayOnce1 = TRUE;
                        dataEEPROM();

                    }

                break;

                case GET_IN_DATA:
                    if(USBHostCDC_Api_Get_IN_Data(MAX_NO_OF_IN_BYTES,
    USB_CDC_IN_Data_Array))
                    {
                        APPL_Demo_State = GET_IN_DATA_WAIT;
                    }
                    else
                    {
                        APPL_Demo_State = READY_TO_TX_RX;
                    }
```

```
                break;

                case GET_IN_DATA_WAIT:

    if(USBHostCDC_ApiTransferIsComplete(&ErrorDriver,&NumOfBytesRcvd))
                    {   /* check for error */
                        if(!ErrorDriver)
                        {
                            if(NumOfBytesRcvd > 0)
                            {
                                analizar();




                            }
                            APPL_Demo_State = READY_TO_TX_RX;

                        }
                        else
                        {
                            putsUART1((unsigned int *)"Host CDC Demo: Error in
    GET_IN_DATA_WAIT\n");
                            APPL_Demo_State = READY_TO_TX_RX;

                        }
                    }
                break;

                case SEND_OUT_DATA:
                    if(USBHostCDC_Api_Send_OUT_Data(sizeof(Rxdata)

    ,USB_CDC_OUT_Data_Array))
                    {
                        APPL_Demo_State = SEND_OUT_DATA_WAIT;
                    }
                    else
                    {
                        APPL_Demo_State = READY_TO_TX_RX; //TODO : Check error

                    }
                break;

                case SEND_OUT_DATA_WAIT:

    if(USBHostCDC_ApiTransferIsComplete(&ErrorDriver,&NumOfBytesRcvd))
                    {
                         ENVIARaHOKUYO = FALSE;
                         USBHostCDC_Clear_Out_DATA_Array();
                         Clear_Rxdata_Array();
                         APPL_Demo_State = READY_TO_TX_RX;
                    }
```

```
                break;

                case CONFIG_SEND_OUT_PC:
                    configCMD();
                    notificar();

                    ENVIANDOaPC = TRUE;
                    APPL_Demo_State = SEND_OUT_PC_WAIT;

                break;

                case SEND_OUT_PC_WAIT:
                    if(ENVIANDOaPC == FALSE)
                    {
                       //USBHostCDC_Clear_In_DATA_Array();
                       APPL_Demo_State = READY_TO_TX_RX;
                    }
                break;

                default:
                break;
            }
        }
    }
```